SONY

MSX 2

GUIDE DU

MSX-BASIC Version 2.0

HITBIT



QUELQUES MOTS D'INTRODUCTION

Cet ouvrage, qui a pour but de vous présenter le MSX2-BASIC, se divise en deux grands volets, à savoir les sections "ETAPES ELEMENTAIRES" et "ETAPES AVANCEES".
La section "ETAPES ELEMENTAIRES" s'adresse à tous ceux qui abordent pour la première fois le langage BASIC. Elle cherche à leur montrer en quoi consiste le BASIC et elle explique la rédaction de programmes simples en BASIC. En vous exerçant à l'écriture de programmes dans la section "ETAPES ELEMENTAIRES", vous ferez connaissance avec toutes les instructions importantes, nécessaires pour la rédaction de programmes en BASIC.
Si vous n'êtes plus un novice en MSX-BASIC, vous pourrez ignorer les "ETAPES ELEMENTAIRES" et passer directement aux "ETAPES AVANCEES". Cependant, si vous n'êtes pas encore familiarisés avec la rédaction de programmes en MSX- BASIC mais que vous disposez d'une certaine formation en un autre langage BASIC, nous vous conseillons de survoler rapidement la section "ETAPES ELEMENTAIRES" afin de faire plus ample connaissance avec les principales instructions du MSX-BASIC, puis de passer à l'étude de la seconde partie, intitulée "ETAPES AVANCEES".

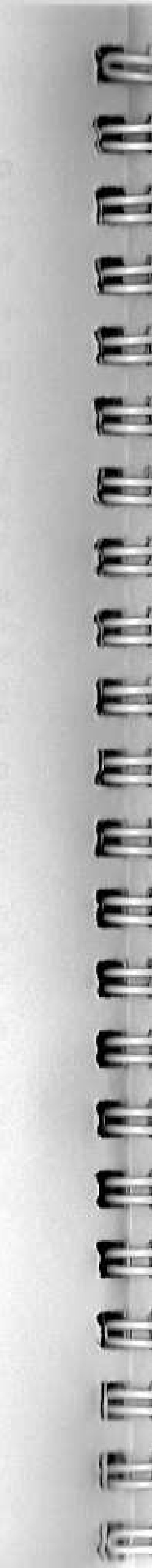
MSX-BASIC Version 2.0

Cet ouvrage explique la façon de rédiger des programmes en MSX-BASIC Version 2.0.
Le MSX-BASIC Version 2.0 est une version plus puissante du MSX-BASIC, utilisé dans les ordinateurs Sony HB-55P, HB-75P/B, HB-101P, et HB-201P. Toutes les instructions et fonctions du MSX-BASIC ont été reprises dans la Version 2.0 et, par conséquent, tout programme, écrit en MSX-BASIC, pourra être exécuté sur un ordinateur qui fait appel au MSX-BASIC Version 2.0.
Dans le présent manuel, les instructions et les fonctions, utilisées exclusivement pour la Version 2.0, seront codifiées comme **MSX2-BASIC**, tandis que nous parlerons simplement de **BASIC** pour toutes les fonctions et instructions qui s'appliquent à la fois au MSX-BASIC et MSX2-BASIC.

TABLE DES MATIERES

ETAPES ELEMENTAIRES

ETAPES AVANCEES



ETAPES ELEMENTAIRES

Chapitre 1
En quoi consiste un programme?

Exécuter l'énoncé suivant en premier.

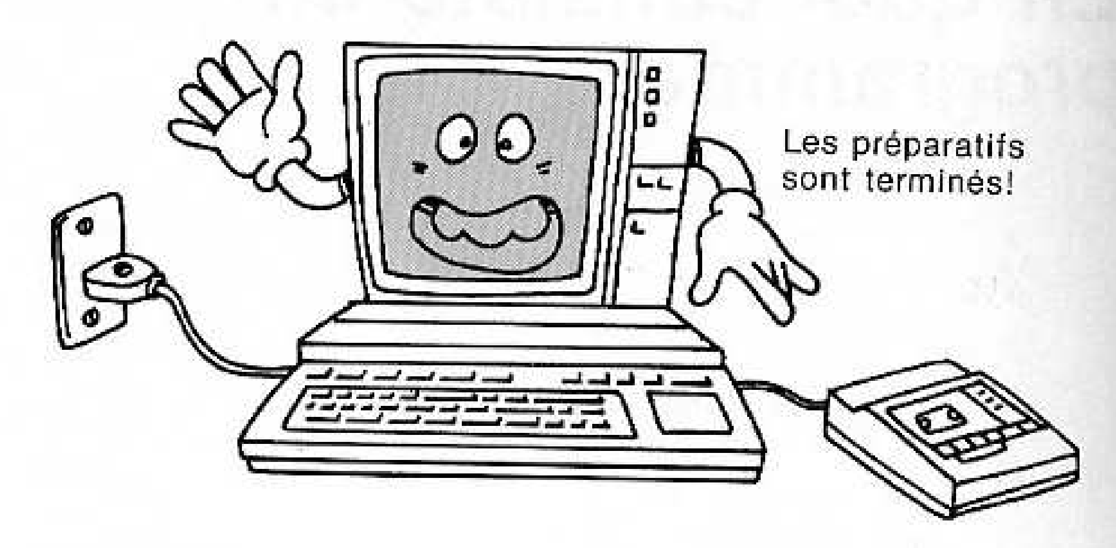
```
SET TITLE "",1
SET PROMPT "OK"
SET ADJUST (0,0)
SET BEEP 1,3
SCREEN 0,,1,1,0,0
KEY ON
WIDTH 39
COLOR 15,4,4
```

VOS PREMIERES INSTRUCTIONS A L'ORDINATEUR

- Comment donner des instructions
- Le MSX-BASIC
- Réalisation de calculs—PRINT "expression"
- Les variables et l'instruction LET
- Affichage de caractères—PRINT "chaîne de caractères"
- Les variables en chaîne

VERIFICATIONS PRELIMINAIRES

Avant tout, veuillez vérifier si vous disposez bien de tout ce qui est requis pour votre étude des programmes BASIC. L'ordinateur et le moniteur de télévision sont-ils raccordés? Dans la négative, consultez à ce sujet le manuel de votre ordinateur. Si ce dernier ne possède pas d'unité à disque souple incorporée, vous devrez raccorder soit un magnétocassette, soit une unité à disque. Ces simples démarches suffisent pour vous mettre au travail.



Remarque

Si votre téléviseur est un appareil couleur sans haut-parleur, vous devrez raccorder un haut-parleur extérieur car le son joue un rôle important dans l'exploitation de votre ordinateur.

MISE EN SERVICE DU BASIC

Quand les préparatifs sont achevés, mettez votre ordinateur sous tension en consultant le manuel qui l'accompagne. Vous y trouverez également les démarches, nécessaires à la mise en marche du BASIC. Après avoir lancé le BASIC, un des affichages suivants apparaîtra sur votre écran.

```
MSX BASIC version 2.0
Copyright 1985 by Microsoft
xxxxx Bytes free
Ok
■
```

Sans unité de disque

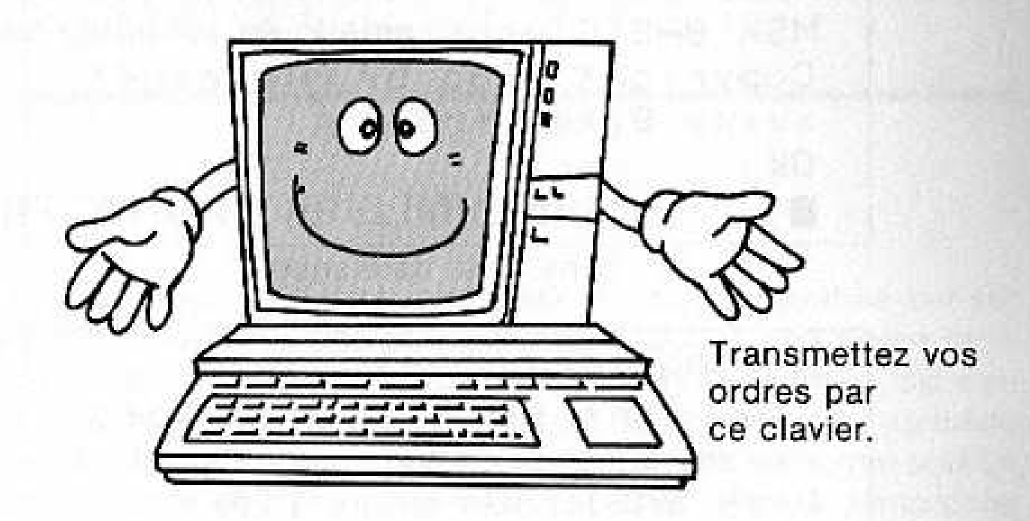
```
MSX BASIC version 2.0
Copyright 1985 by Microsoft
xxxxx Bytes free
Disk BASIC version 1.0
Ok
■
```

Avec une unité de disque

Le BASIC est prêt à fonctionner si, comme illustré ci-dessus, le message "Ok" apparaît à l'avant-dernière ligne, suivi d'un repère carré à la dernière.

UTILISATION DU CLAVIER POUR L'ENTREE DES INSTRUCTIONS

Pour transmettre vos instructions à l'ordinateur, vous utiliserez le clavier de ce dernier. C'est par les lettres et les chiffres que vous taperez sur le clavier que vous ordonnerez à la machine d'effectuer, pour vous, toutes sortes de travaux.



Transmettez vos ordres par ce clavier.

INSTRUCTION 1—"COFFEE PLEASE"

Demandons à notre ordinateur de nous servir un peu de café en tapant les lettres suivantes sur le clavier.

[C][O][F][F][E][E][ ][P][L][E][A][S][E]

Le rectangle blanc [ ] entre les deux mots correspond à une poussée sur la **barre d'espacement**. Chaque fois qu'une touche de clavier est actionnée, le caractère correspondant apparaît sur l'écran à l'endroit où se trouvait le repère carré.
Ce carré est ce que l'on appelle ici le "**curseur**" et il sert à indiquer où sera écrit le caractère suivant.
Si vous avez dactylographié correctement l'instruction "COFFEE PLEASE", l'écran doit se présenter comme suit:

```
Ok
COFFEE  PLEASE ■
```

La démarche suivante est très importante. Après l'entrée de l'instruction, vous devez appuyer sur la touche [↵] (touche de retour). Par cette poussée, vous dites à l'ordinateur que l'entrée de l'instruction est terminée et qu'il doit ensuite l'exécuter.

COFFEE PLEASE

"J'ai fini de donner mes ordres!"

Que se passe-t-il sur l'écran lorsque vous actionnez la touche ↵ ?

```
Ok
COFFEE PLEASE
Syntax error
█
```

Dès l'instant où vous avez actionné la touche ↵ une tonalité "Bip!" s'est faite entendre et le message "Syntax error" est apparu. Ceci constitue la réponse de votre ordinateur à votre ordre "COFFEE PLEASE."
Le message "Syntax error" signifie que vous avez commis une erreur dans la transmission de votre volonté. En effet, l'ordinateur n'est pas capable de comprendre l'ordre que vous lui avez donné.

Le BASIC
Votre ordinateur n'est pas conçu pour comprendre des instructions du genre "COFFEE PLEASE" et, par conséquent, il ne peut, hélas, vous servir une tasse de bon café. Mais il existe bien d'autres ordres qu'il comprend parfaitement et où il excelle. Ce sont les instructions du langage BASIC que vous utiliserez désormais pour lui demander d'effectuer, à votre place, des tâches intéressantes et même très complexes. Considérons maintenant quelques-unes des instructions que l'ordinateur est capable de comprendre.

REALISATION DE CALCULS PRINT expression

Comme pour l'instruction "COFFEE", demandons, par exemple, à l'ordinateur d'ajouter 10 et 5. Entrons les caractères suivants:

[P] [R] [I] [N] [T] [ ] [1] [0] [+] [5] [↵]

L'écran se présentera comme suit:

```
PRINT 10+5
 15
Ok
■
```

L'ordinateur a compris votre instruction PRINT 10+5 et a affiché le résultat, à savoir 15.

PRINT expression

L'instruction PRINT dit à l'ordinateur d'afficher sur l'écran tout ce qui la suit. Il s'agit d'une instruction du langage BASIC que la machine peut parfaitement interpréter. En écrivant une expression mathématique après PRINT, vous demandez, en fait, à l'ordinateur de "faire le calcul et d'en afficher le résultat sur l'écran".

Des calculs variés

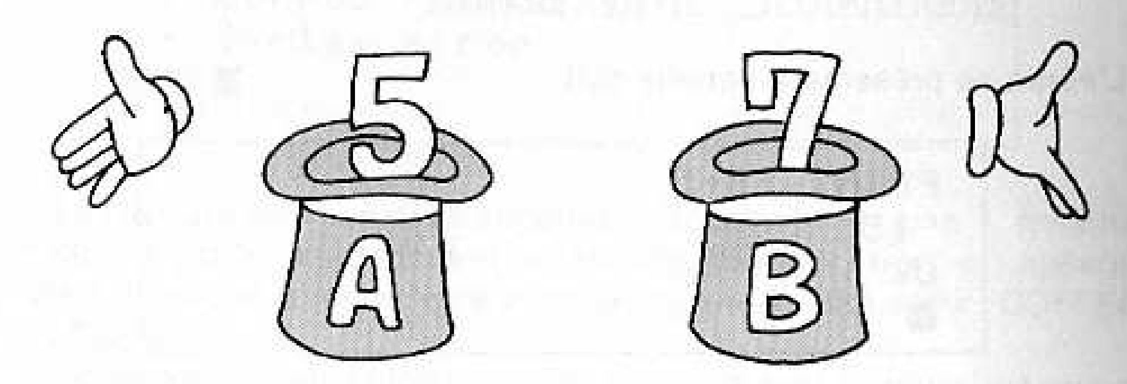
Certes, il ne fallait pas s'équiper d'un ordinateur pour ajouter 5 à 10, mais votre machine est capable d'effectuer aussi des opérations bien plus complexes. Voici d'ailleurs les signes arithmétiques que comprend votre ordinateur.

	Signe	Exemple	Signification
Addition	+	PRINT 123+234	123+234
Soustraction	−	PRINT 300−125	300−125
Multiplication	*	PRINT 9*8	9×8
Division	/	PRINT 72/36	72÷36
Puissance	∧	PRINT 5∧3	5^3

Si vous désirez que l'opération d'une formule soit effectuée en premier lieu, vous pourrez la placer entre parenthèses ( ).

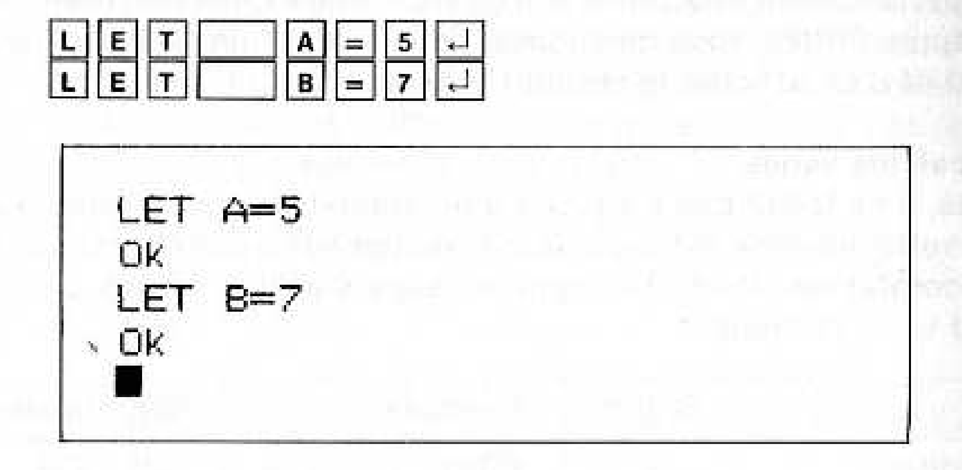
AFFECTATION DE VALEURS A DES VARIABLES LET

Imaginons que nous disposons de deux chapeaux et que nous plaçons un 5 dans le chapeau "A" et un 7 dans le chapeau "B".



Quelle serait la somme A + B? Bien entendu, ce serait 5 + 7 = 12. Quel serait alors le produit de A × B? Cela va de soi: c'est 35.

Effectuons maintenant les mêmes démarches sur l'ordinateur. Vous placez tout d'abord 5 en "A" et 7 en "B". A cet effet, tapez ce qui suit sur le clavier:



L'ordinateur ne répond que par son message "Ok", mais à l'intérieur s'est produite une opération comparable à l'introduction de nombres dans les chapeaux.

LET variable = valeur

LET est l'instruction à utiliser pour placer un nombre dans une variable.

Dans les instructions ci-dessus, les lettres A et B ont pris la place des chapeaux sur les illustrations. Une lettre comme A ou B, utilisée pour contenir un nombre, est appelé une **variable.**
Examinons de nouveau l'instruction précédente.

```
LET A=5
```

La règle à retenir à l'emploi de l'instruction LET est la suivante:

Placer la lettre variable à la gauche du signe égal et le nombre à la droite du signe égal.

Dans ce cas-ci, à la différence des problèmes d'arithmétique, le signe = n'indique pas que les valeurs sont égales. Il vaudrait mieux imaginer qu'il veut dire que "**le nombre de droite entre dans la variable de gauche**".

Nous avons donné deux ordres à l'ordinateur.

```
LET A=5
LET B=7
```

Ainsi donc, 5 a été placé dans A, et 7 l'a été dans B. Utilisons maintenant l'instruction PRINT pour vérifier ce qui s'est passé en réalité.

```
PRINT A
 5
Ok
PRINT B
 7
Ok
```

Voici comment apparaît l'instruction "PRINT expression". Par cet exemple, vous comprenez qu'on peut utiliser des **variables** comme partie de l'expression. Essayons les instructions suivantes:

```
PRINT A+B
 12
OK
PRINT A*B
 35
OK
```

Quand nous avons utilisé seulement des variables dans la partie expression de l'instruction "PRINT expression", l'ordinateur s'est servi des nombres, placés dans les variables, et il nous a fourni les réponses.

Quelques tours de magie à l'aide de l'instruction LET

```
LET C=A*2
OK
PRINT C
 10
OK
```

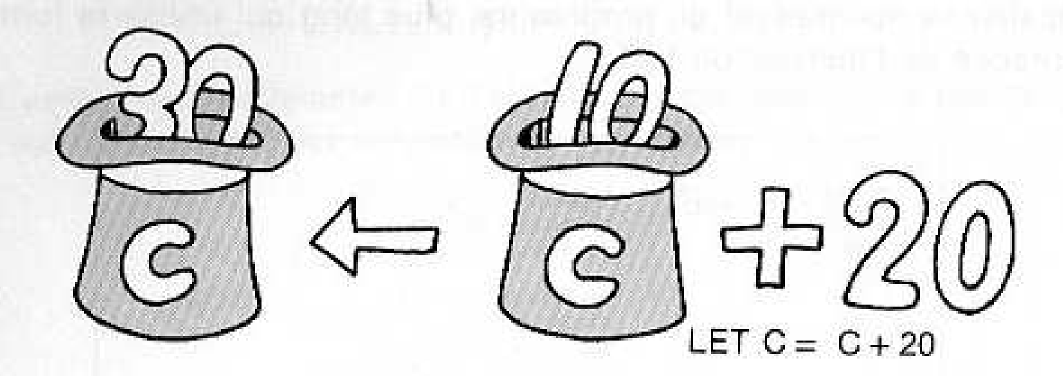
Ici, nous faisons appel à une nouvelle variable, C. A l'aide de l'instruction LET, nous avons fait en sorte que la valeur de C soit la valeur de l'expression A×2. A est une de nos anciennes variables et nous lui avons affecté la valeur 5. Par conséquent, la valeur de C devient 10. Utilisons à présent C pour obtenir une nouvelle valeur.

```
LET C=C+20
OK
PRINT C
 30
OK
```

Regardons le premier énoncé donné à l'ordinateur:

```
LET C=C+20
```

Si vous considérez que le signe = ci-dessus signifie "égal", la formule vous déconcertera certainement. Mais souvenez-vous que **le signe = indique que ce qui se trouve à sa droite devient la valeur de ce qui est placé à sa gauche.** Etant donné que la valeur originale de C était 10, l'expression dit à l'ordinateur d'utiliser cette valeur, puis d'y ajouter 20 et de lui donner la nouvelle valeur qui, en l'occurrence, est 30.



A première vue, ce processus peut sembler un peu fastidieux, mais lorsque vous en avez compris le raisonnement, il devient très simple.

Jusqu'à présent, nous avons étudié l'utilisation de trois variables, à savoir A, B et C et nous pouvons maintenant énoncer certaines règles générales, régissant l'emploi des variables.

REGLES SUR L'EMPLOI DES VARIABLES

- Le nombre de lettres utilisées pour une variable est sans importance, mais l'ordinateur ne considère que les deux premières. Ainsi, des variables comme

  ABC, ABCD, ABD, AB1 ou AB2

  seront toutes considérées par l'ordinateur comme une seule et même variable:

  AB
- Des chiffres ou des signes typographiques ne peuvent pas être utilisés comme premier caractère d'une variable. Des groupes comme A1 ou C3 sont utilisables comme variables, mais 1AB ou *AB ne le sont pas.
- Comme variables, on ne peut pas utiliser des fonctions ou des mots, appartenant au vocabulaire de langage BASIC. En outre, si une partie d'une variable renferme ces termes, l'ordinateur les refusera.

  Ainsi par exemple, des termes comme PRINT, LET, PRINTA ou ALET sont exclus comme variables.

Omission de l'instruction LET

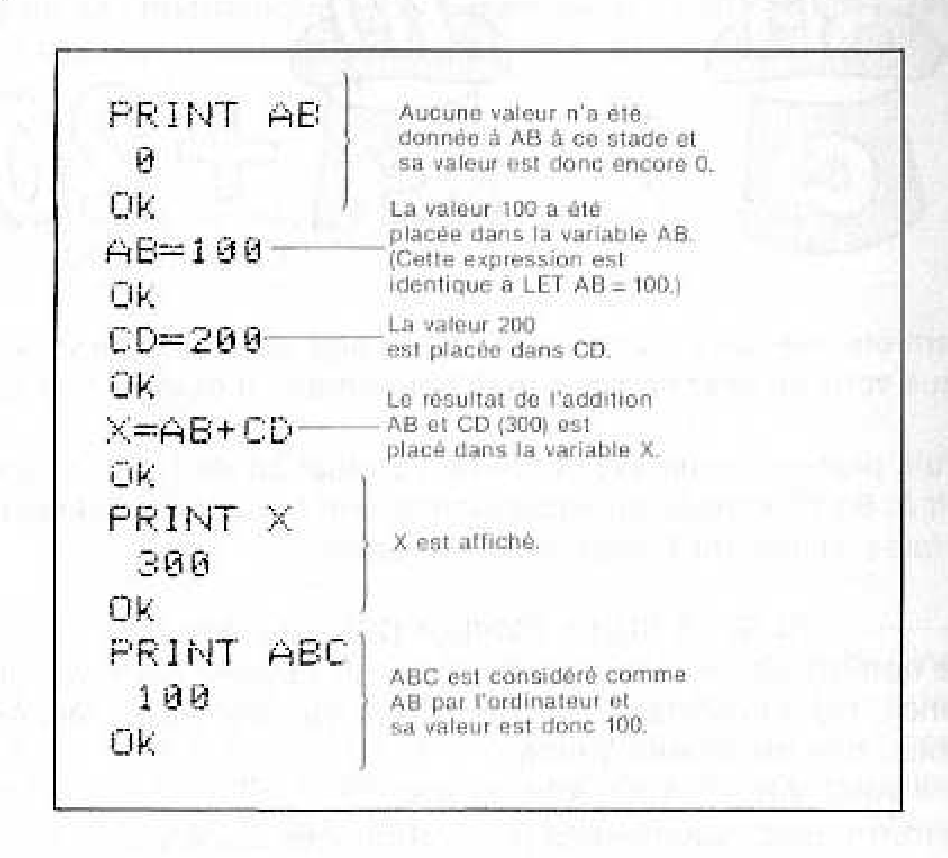
L'emploi très fréquent de variables en langage BASIC entraîne fatalement la répétition de l'instruction LET. Afin de vous faciliter la tâche, le BASIC vous permet cependant d'omettre cette entrée lors de la rédaction d'une variable. Au lieu de

LET A = 10

vous pourrez donc écrire plus simplement:

A = 10

Réalisons maintenant un programme plus long qui utilise la forme abrégée de l'instruction LET.



AFFICHAGE DE CARACTERES

PRINT "chaîne de caractères"

Cette instruction s'emploie pour l'affichage de mots sur l'écran. La même instruction PRINT sert aussi pour les mots. Essayons par exemple ce qui suit:

[P] [R] [I] [N] [T] [ ] ["] [S] [O] [N] [Y] ["] [↵]

L'entrée des guillemets (") s'obtient par poussée sur la touche [2] tout en appuyant sur la touche de majuscule.

```
PRINT "SONY"
SONY
Ok
```

Tout mot ou phrase à imprimer peut être affiché en l'insérant entre des guillemets " ", après l'instruction PRINT. Ci-dessus, le mot "SONY" est contenu entre les guillemets et il sera donc affiché. Les mots compris entre les " " portent le nom de **chaîne de caractères.**

PRINT "chaîne de caractères"

Essayez vous-même de placer certains mots entre guillemets. Par exemple:

```
PRINT "MSX2"
MSX2
Ok
PRINT "3+5"
3+5
Ok
```

Dans la seconde instruction PRINT

```
PRINT "3+5"
```

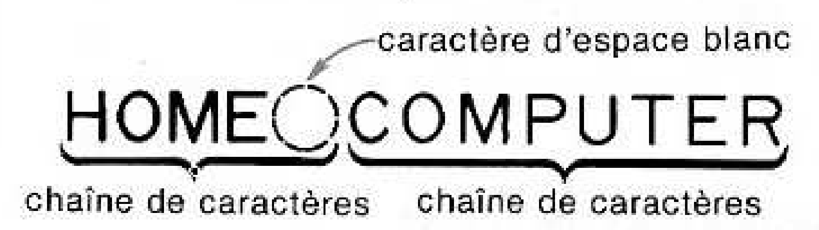
3+5 est aussi une expression arithmétique, mais comme elle est encadrée par des guillemets, elle est considérée comme les trois mots "trois plus cinq" et non pas comme l'expression d'une somme de deux chiffres. Si vous enlevez les guillemets, la somme 3+5 sera calculée et le résultat (8) apparaîtra. Aussi, lors de la rédaction d'un programme pour ordinateur, est-il très important de se souvenir du fait que, **selon la présence ou l'absence des guillemets, des chiffres peuvent servir tantôt comme valeurs numériques, tantôt comme lettres.**

Addition des caractères

Pour ajouter des lettres les unes aux autres, vous utiliserez à nouveau les guillemets et un signe +. Essayez donc ce qui suit.

```
PRINT "SO"+"NY"
SONY
OK
PRINT "HOME "+"COMPUTER"
HOME COMPUTER
OK
```

Dans le premier exemple ci-dessus, les chaînes de caractères "SO" et "NY" sont reliées et affichées comme un seul mot: "SONY." Par contre, dans le second exemple, un espace a été intercalé après la lettre E du mot "HOME". (Cet espace est inséré par une pression sur la barre d'espacement du clavier.) Et c'est parce qu'un espace a été laissé après le E que les deux mots restent séparés lorsqu'ils sont réunis. De ceci, on peut comprendre que même un espace blanc est considéré comme un caractère par l'ordinateur.



A présent que nous pouvons créer des chaînes de caractères comme celle-ci, nous avons besoin de certains chapeaux (variables) pour les y placer.

UTILISATION DU SYMBOLE $ AVEC DES VARIABLES EN CHAINE

Plus haut, nous avons placé des nombres dans des variables, telles que A, B ou CD. Mais rien ne nous empêche de nous servir de variables pour contenir, cette fois, des chaînes de caractères. Dans ce cas, on parlera plutôt de **variables de type en chaîne** (ou simplement de variables en chaîne) pour les distinguer de celles qui renferment des nombres et qui portent, elles, le nom de **variables de type numérique** (ou simplement de variables numériques).
Une variable suivie du symbole $ devient une variable en chaîne, du genre A$, B$ ou CD$. Toute variable suivie du symbole $ n'acceptera qu'une chaîne de caractères comme contenu. (Si un nombre est utilisé comme mot, il peut aussi être placé dans une variable pourvu que le symbole $ y soit attaché.)

Passons à un peu de pratique dans l'utilisation des variables en chaîne.

```
LET A$="SONY"
Ok
PRINT A$
SONY
Ok
```

Ici, nous avons fait appel à l'instruction LET pour placer la chaîne de caractères "SONY" dans la variable en chaîne A$. Rappelons-nous que nous aurions pu omettre le mot LET lors de l'écriture de l'instruction LET. N'oublions pas non plus qu'une chaîne de caractères doit être comprise entre des guillemets " " pour pouvoir être placée dans une variable en chaîne.

```
B$="123"
Ok
PRINT B$
123
Ok
C$=123
Type mismatch
```

Dans l'exemple précédent, nous avons essayé de placer la valeur numérique 123 dans la variable en chaîne C$. Cependant, l'ordinateur nous a répondu en affichant le **message d'erreur "Type mismatch"**, ce qui est sa façon de nous dire que nous avons tenté de faire correspondre deux choses distinctes, en l'occurrence de placer une valeur numérique dans une variable de type en chaîne.

LES MESSAGES D'ERREUR

Si vous donnez à l'ordinateur un ordre qu'il ne comprend pas, il vous répond par un message, exprimant le genre d'erreur commise et appelé **"message d'erreur"**. Si l'ordinateur reçoit une instruction incompréhensible pour lui, il affiche:

```
Syntax error
```

Si vous ne faites pas correspondre les types de valeurs et de variables adéquats, il affiche

```
Type mismatch
```

pour vous signaler quelle erreur vous avez commise.

REALISATION D'UN PROGRAMME EN BASIC

- Un programme d'une ligne
- Vérification d'un programme—LIST
- Exécution d'un programme—RUN
- Effacement d'un programme en mémoire—NEW
- Entrée de valeurs variables par le clavier—INPUT
- Utilisation optimale de l'instruction PRINT
- Effacement d'une partie d'un programme—DELETE
- Effacement de l'affichage sur l'écran—CLS
- Réalisation d'une boucle—GOTO

Dans le chapitre précédent, nous avons appris comment donner des ordres à l'ordinateur. Nous avons vu que, sitôt après avoir introduit une instruction via le clavier et avoir appuyé sur la touche [↵], l'instruction en question était exécutée.
Mais on attend d'un ordinateur qu'il accomplisse successivement de nombreuses opérations. Or, si vous demandez l'exécution d'une instruction chaque fois que vous l'avez exprimée par le clavier, vous freinerez votre démarche car vous ne ferez qu'une opération à la fois. Et c'est ici précisément qu'interviennent les programmes.

LE MODE DIRECT ET LE MODE PROGRAMME

L'entrée d'une simple instruction unique, suivie de son exécution immédiate par une pression sur la touche [↵], comme nous l'avons fait jusqu'ici, est ce que l'on appelle le **mode direct.** Mais il existe, heureusement, un autre et meilleur moyen d'employer un ordinateur et l'on parle alors du **mode programme.** A vrai dire, si l'on n'utilise pas le mode programme, l'ordinateur n'a guère de raison d'être.

UN PROGRAMME D'UNE LIGNE

Utilisons maintenant le mode programme pour créer un programme, d'autant plus que ceci ne requiert rien d'autre que l'entrée des caractères suivants.

[1] [0] [   ] [P] [R] [I] [N] [T] [   ] [3] [+] [5] [↵]

```
10 PRINT 3+5
■
```

L'instruction PRINT 3+5 signifie que l'ordinateur doit additionner 3 et 5 et afficher le résultat, 8. Cependant, rien n'a été affiché quand vous avez appuyé sur la touche [↵] après l'entrée de l'instruction. La raison en est que vous avez entré le nombre 10 avant l'instruction PRINT. Chaque fois qu'une instruction est précédée d'un nombre, l'ordinateur comprend qu'il doit placer cette instruction dans sa mémoire au lieu de l'exécuter lorsque la touche [↵] est actionnée. Ce genre de nombre est appelé un **numéro de ligne**.



Quand la touche [↵] est actionnée, le programme est mémorisé dans l'ordinateur.

L'instruction, précédée d'un numéro de ligne et mémorisée par l'ordinateur, est ce que l'on appelle un **programme**.

VERIFICATION D'UN PROGRAMME LIST

Nous pouvons vérifier si l'ordinateur se souvient bien du programme

```
10 PRINT 3+5
```

en nous servant de l'instruction:

LIST

Entrons les caractères suivants via le clavier.
L I S T ↵

```
LIST
10 PRINT 3+5
Ok
```

L'instruction LIST, qui s'entre en mode direct, c'est-à-dire non précédée d'un numéro de ligne, **sert à demander à l'ordinateur d'afficher le programme.**

EXECUTION D'UN PROGRAMME RUN

En mode direct, on s'est servi de la touche [↵] pour exécuter une instruction. Par contre, en mode programme, c'est l'instruction RUN qui est utilisée.

[RUN]

Entrez les caractères suivants.

[R] [U] [N] [↵]

```
RUN
 8
Ok
```

Le programme qui a été mémorisé par l'ordinateur

```
10 PRINT 3+5
```

est maintenant exécuté pour la première fois et la réponse, 8, est affichée sur l'écran. Si l'on renouvelle l'instruction RUN, le résultat 8 est à nouveau affiché, un processus qui peut se répéter à volonté.

```
RUN
 8
Ok
RUN
 8
Ok
RUN
 8
Ok
```

Qu'est-ce qu'un programme?
Procédons à une comparaison de l'exécution de

```
PRINT 3+5
```

selon le mode direct et le mode programme.

	Mode direct	Mode programme
N° de ligne	Non	Oui
Exécution	Appuyer sur ⏎	Entrer l'instruction RUN
Mémorisation de l'instruction	Non	Oui
Nombre d'exécutions	1 fois seulement	Sans limite

C'est parce qu'un programme comporte des numéros de ligne que l'ordinateur est capable de le mémoriser et de l'exécuter autant de fois qu'on le désire, moyennant l'utilisation de l'instruction RUN.
En fait, sans programme, un ordinateur ne serait qu'une machine stupide.
Un programme est donc une procédure, écrite afin de faire exécuter par un ordinateur une série d'opérations plus ou moins complexes et l'on parle, dans ce cas, de **logiciel**. Par contre, pour désigner l'ordinateur proprement dit, une machine conçue pour accomplir fidèlement une série donnée d'actions, il est convenu de parler de **matériel**.

EFFACEMENT D'UN PROGRAMME EN MEMOIRE NEW

Nous souhaitons maintenant écrire un nouveau programme et le faire mémoriser par l'ordinateur, mais la mémoire de ce dernier est encore occupée par le programme:

```
10 PRINT 3+5
```

Par conséquent, nous devons exécuter l'instruction NEW de sorte que l'ordinateur puisse se souvenir de notre nouveau programme.

NEW

L'instruction NEW a pour mission d'effacer la totalité du programme qui se trouvait dans la mémoire de l'ordinateur.

Entrez l'instruction NEW et servez-vous ensuite de l'instruction LIST afin de vérifier si le programme a bel et bien été effacé.

N E W ↵

```
NEW
OK
LIST
OK
```

Comme on le constate, le programme n'est plus affiché.

Les façons de vider la mémoire

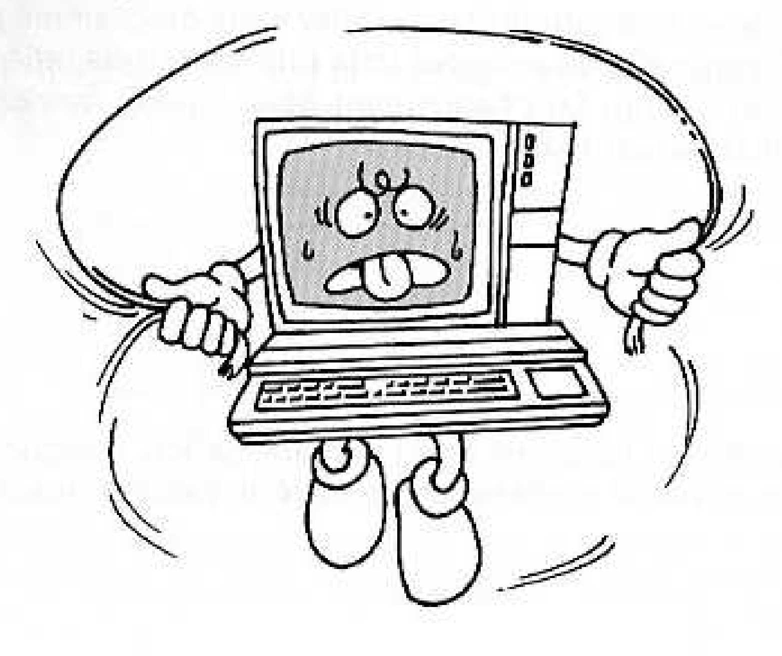
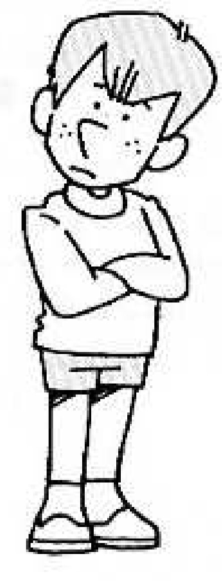
Vous pourrez effacer un programme, placé dans la mémoire de votre ordinateur, en faisant appel à l'instruction NEW. Mais le même résultat sera obtenu dans les cas suivants.

- Vous appuyez sur la touche [RESET].
- L'ordinateur est mis hors tension.

La touche [RESET] est prévue pour les cas d'urgence, comme celui où une erreur s'est glissée dans un de vos programmes, rendant impossible l'arrêt de son exécution, quelle que soit la touche actionnée.

Une poussée sur la touche [RESET] équivaut à une mise hors tension de l'appareil, suivie d'une nouvelle mise sous tension. Par conséquent, cette action efface le programme en mémoire et rétablit l'ordinateur dans l'état où il se trouvait lors de sa mise sous tension originale.

Par conséquent, on prendra garde à ne pas actionner la touche [RESET] ou l'interrupteur d'alimentation au cours de l'élaboration d'un programme.

ENTREE DE VALEURS VARIABLES PAR LE CLAVIER INPUT

Essayons maintenant de modifier ce programme de manière qu'à chaque exécution, nous puissions affecter une valeur différente à A et à B. Si nous faisons ceci, chaque fois que le programme sera exécuté, des nombres différents seront additionnés. A cet effet, nous devrons faire appel, non pas à l'instruction LET, mais à une autre, l'instruction INPUT.

INPUT nom de variable

INPUT est l'instruction à utiliser pour entrer une valeur dans une variable par l'intermédiaire du clavier.

```
10 INPUT A
20 INPUT B
30 PRINT A+B
```

Une fois que ces lignes sont introduites, vérifiez votre programme par l'instruction LIST, comme vous en aurez déjà pris l'habitude, afin de confirmer sa mémorisation. Le changement étant opéré, lancez le programme par l'instruction RUN.

```
RUN
? ■
```

Dès que le programme est lancé, un point d'interrogation, flanqué du curseur à sa droite, apparaît au-dessous de RUN. Il s'agit du résultat de l'instruction

```
INPUT A
```

de la ligne 10. L'ordinateur attend que, par le clavier, vous lui disiez, dans une instruction, quelle valeur il doit placer dans la variable A. N'importe quel nombre ferait l'affaire; essayons donc 25.

[2] [5] [↵]

Souvenez-vous d'actionner la touche [↵] après avoir tapé 25. Vous signifiez ainsi à l'ordinateur que 25 est le nombre qu'il doit placer dans la variable A.

```
RUN
? 25
? ■
```

Après avoir entré 25, le point d'interrogation apparaît de nouveau. Cette fois, c'est le résultat de l'exécution de la ligne 20 par la machine. Entrons 75 pour la variable B.

[7] [5] [↵]

Une fois que 25 est entré pour A et 75 pour B, la ligne 30 est exécutée et 100, c.à.d. le total de A + B, est affiché.

```
RUN
? 25
? 75
 100
Ok
```

Lancez à nouveau ce programme en donnant des valeurs différentes à A et à B et vous constaterez que le résultat exact est toujours affiché à l'écran.

Pour rendre l'instruction INPUT plus "facile d'emploi"
Au cours de l'exécution de ce programme, la seule chose visible sur l'écran est un point d'interrogation, accompagné du curseur. Si vous êtes vous-même l'auteur du programme, vous saurez, bien entendu, qu'il convient, à ce stade, d'entrer la valeur pour la variable A. Mais tout autre utilisateur restera sans réponse devant ce point d'interrogation. Afin de rendre ce programme plus "facile d'emploi" pour autrui, nous pouvons modifier comme suit l'instruction INPUT.

```
10 INPUT "A=";A
20 INPUT "B=";B
30 PRINT A+B
```

Lorsque vous exécuterez le programme, après y avoir apporté les changements ci-dessus, l'écran se présentera d'abord comme suit:

```
RUN
A=? ■
```

La chaîne de caractères, placée entre des guillemets, avant le nom de variable est appelée un **énoncé de signalisation**; celui-ci reste affiché devant le point d'interrogation pendant le déroulement du programme. L'emploi d'un tel énoncé de signalisation permet donc à quiconque de comprendre ce qui doit être entré.

INPUT "énoncé de signalisation"; nom de variable

N'oubliez pas qu'un point-virgule (;) doit être inséré entre l'énoncé de signalisation et le nom de variable.

Faisons passer tout le programme.

```
RUN
A=? 1234
B=? 4321
 5555
OK
```

N'importe quels mots peuvent servir comme énoncé de signalisation. Ainsi par exemple:

```
10 INPUT "First value";A
20 INPUT "Second value";B
30 PRINT A+B
```

UTILISATION OPTIMALE DE L'INSTRUCTION PRINT

Grâce aux changements apportés au programme ci-dessus, notre machine commence à se mieux comporter à la façon d'un ordinateur. Toutefois, il est encore possible de faciliter l'utilisation de ce qui apparaît à l'écran. Pour l'instant, le programme affiche seulement la somme de A et de B, mais il ne nous dit pas ce qu'elle signifie. Ne vaudrait-il pas mieux que l'ordinateur nous dise que le nombre affiché est le résultat de A + B? Cette information peut s'obtenir par l'instruction PRINT, placée dans une ligne ajoutée au programme, comme ci-après.

```
10 INPUT "A=";A
20 INPUT "B=";B
25 PRINT "A+B="  ←— ligne ajoutée
30 PRINT A+B
```

Ici, la ligne 25 a été ajoutée au programme; ceci peut se faire à tout moment quand "Ok" est affiché. Vous taperez donc cette ligne exactement comme vous le feriez pour toute autre dans un nouveau programme.

[2] [5] [ ] [P] [R] [I] [N] [T] [ ] ["] [A] [+] [B] [=] ["] [↵]

Entrez alors l'instruction LIST pour confirmer que la ligne 25 a été ajoutée au programme.

```
LIST
10 INPUT "A=";A
20 INPUT "B=";B
25 PRINT "A+B="
30 PRINT A+B
OK
```

La ligne 25 se trouve bel et bien insérée entre les lignes 20 et 30, sa place normale. Vous n'avez pas à vous préoccuper de l'ordre d'entrée des lignes du programme car l'ordinateur se chargera lui-même de les arranger dans l'ordre croissant.

Lorsque vous avez introduit la ligne 25, vous aurez sans doute compris pourquoi les nombres 10, 20 et 30 avaient été utilisés comme numéros de ligne. Vous l'aviez deviné! C'était pour laisser de l'espace, en vue d'insertions ultérieures.

Quand vous aurez ajouté la ligne 25, l'ordinateur affichera "A + B = " avant d'afficher la somme de A et B. Faisons maintenant passer ce programme et examinons à quoi il ressemble.

```
RUN
A=? 100
B=? 50
A+B=          ←—— le résultat de la ligne 25
 150
```

"A + B = " est désormais affiché avant la réponse 150, mais ne serait-ce pas plus clair si A + B = et 150 étaient affichés sur la même ligne, comme ceci?

```
A+B= 150
```

Utilisation du point-virgule (;) pour relier deux parties affichées
Modifions encore la ligne 25 comme ci-dessous:

```
25 PRINT "A+B=";
                ↑——— ceci a été ajouté
```

Un point-virgule (;) a donc été ajouté après le second guillemet. Effectuez les démarches suivantes pour procéder à une révision. Entrez tout d'abord l'instruction LIST pour afficher la ligne 25.

```
LIST 25
```

N'oubliez pas d'appuyer sur la touche [↵] après avoir tapé l'instruction.

```
LIST 25
25 PRINT "A+B="
Ok
■
```

LIST numéro de ligne

En tapant un numéro de ligne après l'instruction LIST, il est possible d'afficher uniquement la ligne qui y correspond. De même, si vous tapez

LIST numéro de ligne—numéro de ligne

vous afficherez seulement une partie du programme. Par exemple,

```
LIST 20-30
```

affichera les lignes 20 à 30 dans un programme donné. Dans notre exemple précédent, les lignes 20, 25 et 30 seraient donc affichées. Procédons à un essai et examinons le résultat.

Après avoir affiché la ligne 25 par l'instruction LIST, servez-vous des touches de déplacement du curseur (⌖) pour amener le curseur (le repère □) à la droite du dernier caractère (") de la ligne 25, car c'est là qu'il faudra ajouter le point-virgule.

```
25 PRINT "A+B="■
Ok             ↑
        amener ici le curseur
```

Appuyez alors sur la touche [;].

```
25 PRINT "A+B=";■
```

Le point-virgule se trouve affiché sur l'écran, mais cela ne suffit pas. Pour valider le changement, vous devez encore appuyer sur la touche [↵]. Lors de la rédaction du programme original, vous aviez actionné la touche [↵] après chaque ligne. Quand vous révisez une ligne quelconque, cette touche doit de nouveau être actionnée car c'est à cette condition que la ligne modifiée est placée dans la mémoire de l'ordinateur.
Servez-vous de l'instruction LIST pour confirmer la modification apportée à la ligne 25, puis lancez le programme par l'instruction RUN. Le fait d'ajouter le point-virgule a permis d'afficher le résultat sur la même ligne que A + B =.

```
RUN
A=? 100
B=? 50
A+B= 150
Ok
```

Utilisation d'une virgule (,) au lieu d'un point-virgule (;)
Substituons à présent le point-virgule par une virgule à la ligne 25.

```
25 PRINT "A+B=",
```

↑ placer la virgule ici

Après avoir remplacé le point-virgule par une virgule, lancez le programme.

```
RUN
A=? 100
B=? 50
A+B=           ␣150
```

espace de 14 caractères — le 15ème espace à compter du début de la ligne

A présent, le résultat 150 est placé 16 caractères (le 15ème espace est réservé au signe moins éventuel) après le début de la ligne et non plus juste à la droite de A + B =.

EFFACEMENT D'UNE PARTIE D'UN PROGRAMME DELETE

Pour le moment, les lignes 25 et 30 de notre programme sont:

```
25 PRINT "A+B=",
30 PRINT A+B
```

Mais nous pouvons combiner ces deux lignes en une seule.

```
25 PRINT "A+B=",A+B
```

Révisez votre programme pour combiner les deux lignes, comme vous l'avez fait pour remplacer le point-virgule par une virgule.

De ce fait, la ligne 30 est devenue inutile et nous pouvons donc la supprimer. A cet effet, deux méthodes sont disponibles. L'une consiste à utiliser l'instruction DELETE.

```
DELETE 30
```

L'instruction DELETE permet d'effacer des lignes spécifiées d'un programme. La formulation habituelle de cette instruction est:

DELETE N° de ligne-N° de ligne

Après avoir tapé DELETE, spécifiez les numéros de la première et de la dernière ligne de la partie du programme à effacer, en les séparant par un trait d'union (-).
Quand on désire effacer une seule ligne d'un programme, l'instruction DELETE n'est pas requise. En effet, il suffit de taper:

[3] [0] [↵]

EFFACEMENT DE L'AFFICHAGE SUR L'ECRAN CLS

Tel que notre programme est actuellement rédigé, les lignes avancent sur l'écran et le résultat est affiché chaque fois que le programme est exécuté. Les lignes, affichées lors de l'exécution précédente, restent également sur l'écran. Les travaux seraient facilités et plus clairs si l'écran était vidé avant chaque calcul. C'est à cela que sert **l'instruction CLS,** une abréviation de "clear screen" ou "vider l'écran".

CLS

Ajoutons l'instruction CLS à notre programme et plaçons-la à la ligne 5.

```
LIST
5 CLS
10 INPUT "A=";A
20 INPUT "B=";B
25 PRINT "A+B=",A+B
```

Par insertion de la ligne 5 au programme, l'écran est vidé après chaque exécution, ce qui permet de mieux se rendre compte de ce qui s'y passe.

REALISATION D'UNE BOUCLE GOTO

Des boucles par l'instruction GOTO
Et c'est ici qu'intervient une nouvelle instruction, ajoutée au programme.

```
30 GOTO 10
```

Exécutons le programme et observons ce qui se passe après avoir ajouté la nouvelle ligne.

GOTO N° de ligne

L'instruction GOTO enjoint à l'ordinateur d'aller (ou de retourner) au numéro de ligne spécifié.

Normalement, le programme est exécuté en commençant par le plus petit numéro de ligne, mais l'instruction GOTO permet de bouleverser cette convention.

Par conséquent, l'instruction

```
GOTO 10
```

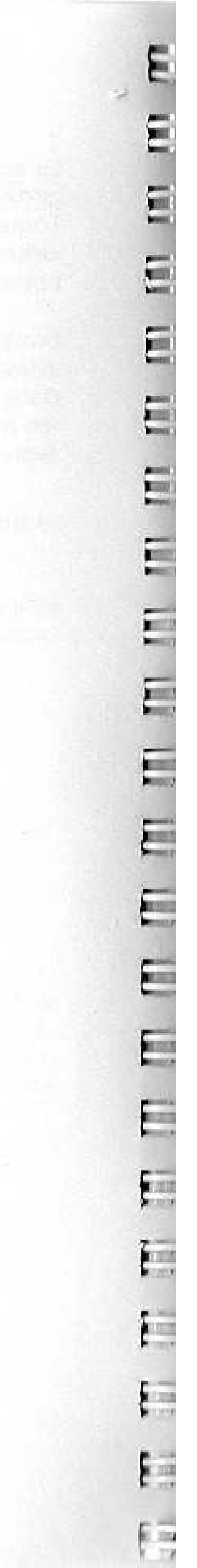
ramène le déroulement du programme à la ligne 10, et il se poursuit à partir de celle-ci. Ainsi, chaque fois que l'on désire voir l'ordinateur sauter à une partie différente du programme, il suffira de spécifier le numéro de ligne de destination par l'instruction GOTO.

Si le déroulement du programme se poursuit, c'est parce que l'instruction, GOTO 10 de la ligne 30 oblige l'ordinateur à repasser à la ligne 10; après avoir exécuté successivement les lignes 10, 20 et 25, il arrive à nouveau à la ligne 30 qui le force, une nouvelle fois, à repasser à la ligne 10, et ainsi de suite, indéfiniment. Ainsi, le programme se déroule dans l'ordre suivant:

```
┌→10
│  ↓
│  20
│  ↓
│  25
│  ↓
└─30
```

La seule façon d'arrêter cette répétition est d'appuyer sur CTRL + STOP.
Toute partie répétée d'un programme s'appelle une **boucle**; quand cette répétition se poursuit indéfiniment, on parle de **boucle sans fin**, obtenue par l'instruction GOTO.

Dans ce premier chapitre, nous vous avons introduit les programmes et avons donné certaines notions, relatives à leur fonctionnement. Dans la section suivante, nous ferons connaissance avec de nouvelles instructions et passerons à la rédaction de programmes plus évolués.



Chapitre 2
A la façon d'un vrai ordinateur

POUR SORTIR D'UNE BOUCLE

- Pour satisfaire à une condition—IF—THEN
- Terminaison d'un programme—END
- Renumérotation des lignes—RENUM

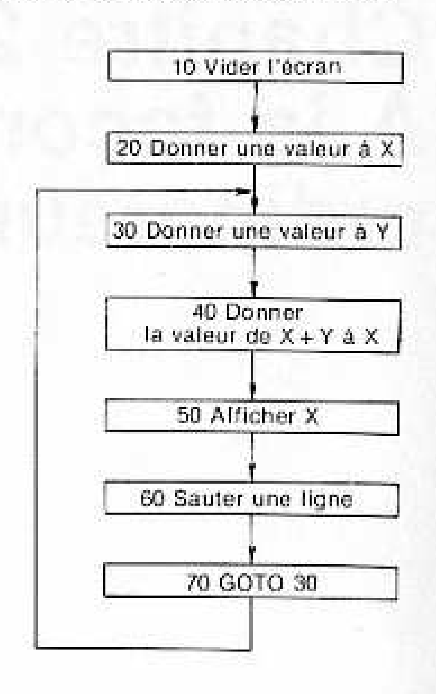
POUR SATISFAIRE A UNE CONDITION IF—THEN

La boucle sans fin

Quand un programme à boucle sans fin est lancé par l'instruction RUN, il se répète normalement sans interruption jusqu'à ce que vous daigniez l'arrêter par une poussée simultanée sur les touches CTRL + STOP. En effet, si vous sortez d'une boucle sans fin par pression sur les touches CTRL + STOP, tout le programme s'arrête. Heureusement, grâce à la méthode que nous allons découvrir, l'exécution du programme se poursuivra, même après être sorti de la boucle sans fin.

Préparons tout d'abord un simple programme à boucle sans fin.

```
10 CLS
20 INPUT "START";X
30 INPUT "ANY NUMBER";Y
40 X=X+Y
50 PRINT "TOTAL=";X
60 PRINT
70 GOTO 30
```



Arrêt du programme sur la base de la valeur de Y

Dans notre vie quotidienne, nous accomplissons toutes sortes d'actions en nous basant sur certaines conditions. Par exemple, s'il fait beau, nous allons jouer au football, mais s'il pleut, la rencontre est annulée. Ou encore, si j'ai de l'argent, je vais voir un bon film, mais si je suis "fauché", je reste chez moi pour faire une sieste.

Nous avons tous rencontré de pareilles situations où nous allons faire une chose si une telle condition est remplie, mais où nous opterons pour une autre activité si la condition est différente.

Qu'en est-il de l'ordinateur? En fait , lui aussi est capable d'agir en se basant sur ce genre de décisions conditionnelles. L'instruction **IF—THEN** permet de faire exécuter l'instruction suivante par l'ordinateur, pourvu qu'une certaine condition soit remplie. En réalité, ce type d'instruction, qui transmet un ordre à l'ordinateur, est normalement appelé un "**énoncé**", comme dans "**énoncé IF—THEN**". Des instructions, telles que **PRINT**, **LET** ou **INPUT**, servent toutes à effectuer un énoncé et, à leur sujet, on parle habituellement **d'énoncé PRINT, d'énoncé LET**, etc.

IF formule conditionnelle THEN énoncé

Le couple **IF—THEN** dit à l'ordinateur que, si la formule conditionnelle suivant **IF** est vraie, il doit alors exécuter l'énoncé qui vient après **THEN**.

Ajoutons ici l'énoncé **IF—THEN** au programme que nous avons déjà écrit.

```
35 IF Y=0 THEN END
```

```
LIST
10 CLS
20 INPUT "START";X
30 INPUT "ANY NUMBER";Y
35 IF Y=0 THEN END
40 X=X+Y
50 PRINT "TOTAL=";X
60 PRINT
70 GOTO 30
```

On comprend aisément le sens de la ligne 35, ajoutée ici. Elle veut dire: "si Y est 0, alors mettre fin au programme".

TERMINAISON D'UN PROGRAMME END

A la ligne 35, la formule conditionnelle est Y = 0, et END est l'énoncé.

END

END est donc l'énoncé, utilisé pour mettre fin au programme. Lorsque cet énoncé est exécuté, le programme s'arrête à ce point.

FORMULES CONDITIONNELLES

La tableau suivant indique quels symboles sont utilisables pour exprimer des **formules conditionnelles.**

Symbole	Signification	Exemple	
=	égal à	IF A=B	Si A est égal à B
>	supérieur à	IF A>B	Si A est supérieur à B
<	inférieur à	IF A<B	Si A est inférieur à B
>= =>	supérieur ou égal à	IF A>=B IF A=>B	Si A est supérieur ou égal à B
<= =<	inférieur ou égal à	IF A<=B IF A=<B	Si A est inférieur ou égal à B
<> ><	différent de	IF A<>B IF A><B	Si A est différent de B

Le symbole utilisé dans la formule conditionnelle de la ligne 35 était

```
IF Y=0
```

ce qui signifie "si Y est égal à 0". Si cette condition est vérifiée, alors l'énoncé qui suit THEN sera exécuté. Dans notre programme, il s'agissait de l'énoncé END et c'est pourquoi le programme s'est arrêté.

Rappel

Dans l'énoncé LET, le signe = avait servi pour indiquer que le nombre placé à sa droite était la valeur du nom de variable, situé à sa gauche. Remarquez, cependant, que quand il est employé dans la formule conditionnelle de l'énoncé IF—THEN, il signifie "égal à".

Examinons un peu plus attentivement comment se déroule le programme.

```
RUN
START? 10
ANY NUMBER? 20
TOTAL= 30

ANY NUMBER? 15
TOTAL= 45

ANY NUMBER? 100
TOTAL= 145

ANY NUMBER? 0
Ok
```

Tout d'abord, 20 a été ajouté à 10, puis 15 a été ajouté à ce total; ensuite, 100 a été ajouté à ce résultat. Lorsqu'un chiffre a été ajouté ensuite, il s'agissait de 0 et, à ce stade, le programme se trouvait à la ligne 30. Comme 0 a été entré comme valeur de la variable Y, à la ligne 35, la formule conditionnelle Y=0 devient vraie et l'énoncé END, placé après THEN, est exécuté et le programme s'arrête.

RENUMEROTATION DES LIGNES RENUM

Au début de la rédaction de notre programme, nous avons utilisé les numéros de lignes en dizaines: 10, 20, 30 ... 70. Par la suite, nous avons été amenés à ajouter une ligne comme 35. Nous aurions pu, tout aussi bien, employer des nombres tels que 1, 3, 28, 29, 78, ou 105 pour numéroter les lignes de notre programme. Mais par souci de clarté, vous pourrez toujours faire appel à l'instruction RENUM pour rétablir les lignes, numérotées par dizaines, au cas où vous auriez ajouté tellement de lignes entre les originales qu'il n'y subsiste plus d'espace.

Entrons donc l'instruction

```
RENUM
```

en mode direct et observons si chaque ligne est bien re-numérotée par multiple de 10.

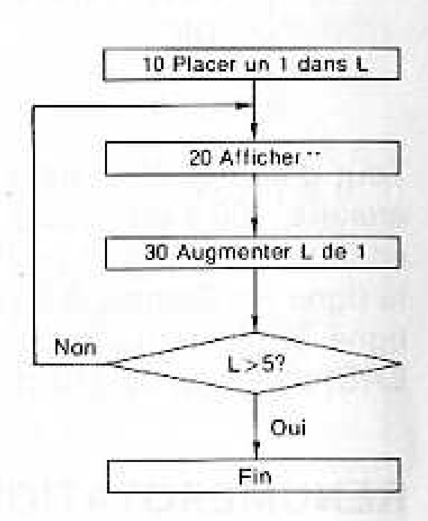
LE SPECIALISTE DES BOUCLES

- Réaliser une boucle—FOR—NEXT
- Une boucle à l'intérieur d'une boucle

SPECIFICATION DE REPETITIONS DE BOUCLES FOR—NEXT

Auparavant, dans ce manuel, nous avons vu comment sortir d'une boucle à l'aide d'un énoncé IF—THEN.
Mais il existe un autre moyen, souvent plus pratique, de spécifier le nombre de répétitions attendues d'une boucle. Etudions cette méthode en nous servant d'un programme simple.

```
10 FOR L=1 TO 5
20 PRINT "** ";
30 NEXT L
```



Quand ce programme est lancé, l'affichage suivant s'inscrit sur l'écran.

```
RUN
** ** ** ** **
Ok
```

L'énoncé PRINT de la ligne 20 de ce programme affiche ** ␣, une fois (␣ indiquant un espace).
Ensuite, par l'exécution de tout le programme, ** ␣ est affiché cinq fois. Ceci provient du fait que la ligne 20 est répétée cinq fois. Les énoncés de la ligne 10 et de la ligne 30 donnent les instructions pour la répétition de la ligne 20.

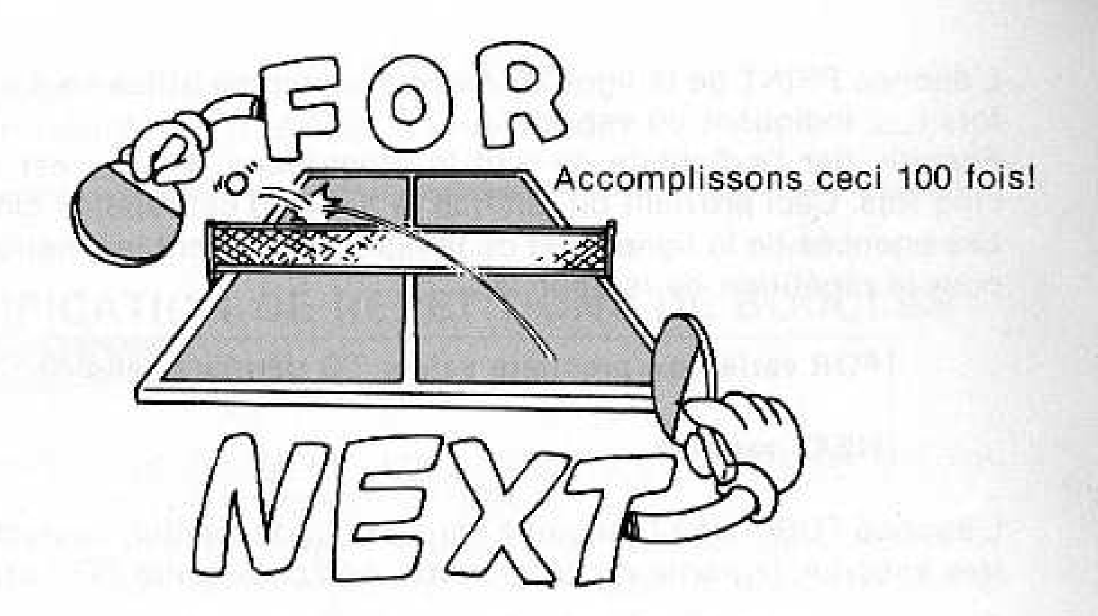
FOR variable = première valeur TO dernière valeur | **NEXT variable**

L'énoncé FOR—NEXT enjoint à l'ordinateur de répéter, du nombre de fois spécifié, la partie du programme, comprise entre FOR et NEXT.

```
FOR L=1 TO 5


NEXT L
```

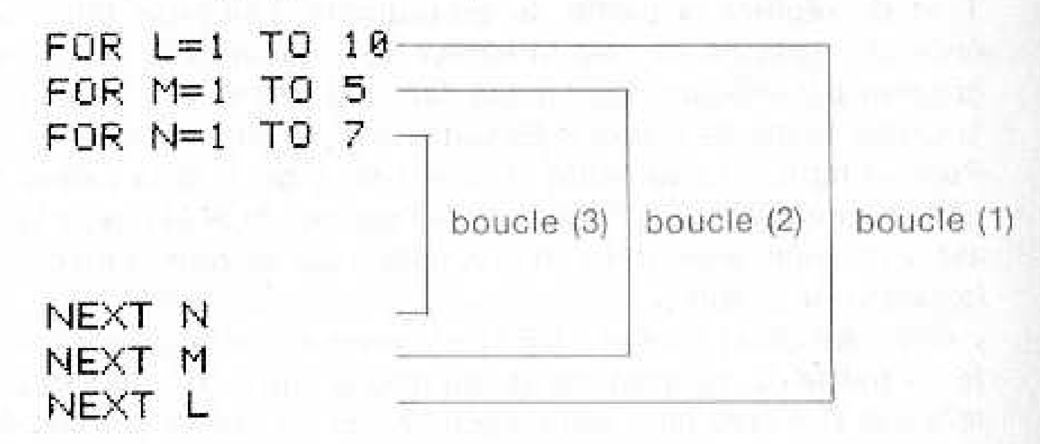
La ligne 10, FOR L = 1 TO 5, et la ligne 30, NEXT L, forment une paire qui dit à l'ordinateur d'augmenter la valeur de L, en commençant par 1, et de répéter la partie du programme, comprise entre les deux énoncés, jusqu'à ce que la valeur de L arrive à 5. Lorsque le programme est exécuté, 1 est placé dans L. La ligne 10 signale aussi que la valeur finale de L sera 5. Ensuite, la ligne 20 est exécutée une fois. Puis, la ligne 30 augmente la valeur de L de 1. Si la valeur de L n'a pas encore atteint 5, la ligne après l'énoncé FOR (à savoir la ligne 20) est à nouveau exécutée, un processus qui se répète jusqu'à ce que la valeur de L soit 5.
L'énoncé FOR et l'énoncé NEXT agissent de cette façon, afin de répéter la partie du programme qui se trouve entre eux, et ceci autant de fois que l'on aura bien voulu spécifier. Etant donné que ces deux éléments sont souvent utilisés comme une paire, on parle habituellement, à leur sujet, de l'énoncé FOR—NEXT, une boucle formée par un énoncé FOR—NEXT portant le nom de **boucle FOR—NEXT** . Un point capital à ne pas oublier, c'est que **la même variable doit être utilisée**, tant pour l'énoncé FOR que pour l'énoncé NEXT (dans ce cas précis, il s'agissait de L).



Une boucle à l'intérieur d'une boucle

L'énoncé FOR—NEXT permet de se livrer à une foule de choses intéressantes. Un exemple? Vous pourrez placer une ou plusieurs autres boucles FOR—NEXT à l'intérieur de la boucle FOR—NEXT qui est comprise dans le premier énoncé FOR—NEXT.

La structure d'un tel programme se présente comme suit:



```
FOR L=1 TO 10
FOR M=1 TO 5
FOR N=1 TO 7

NEXT N
NEXT M
NEXT L
```


Dans cet exemple, la boucle (2) se trouve à l'intérieur de la boucle (1) et la boucle (3) est comprise à l'intérieur des deux précédentes. La boucle (1) est répétée dix fois, mais chaque fois qu'elle se répète, la boucle (2), comprise à l'intérieur de la boucle (1) est elle-même répétée cinq fois. De plus, chaque fois que la boucle (2) se répète, la boucle (3), qui se trouve à l'intérieur, est répétée sept fois.

Lors de la rédaction d'un programme de ce genre, vous devrez toujours utiliser une variable différente (telle que L, M et N ci-dessus) à l'intérieur de chaque boucle.

LES ORDINATEURS ET LEUR LANGAGE

Un ordinateur enregistre vos directives comme une série de démarches et il les exécute dans l'ordre que vous avez défini. La méthode utilisée pour rédiger cette suite d'étapes, cette procédure, est appelée un **langage-machine**. Le BASIC est l'un de ces langages d'informatique, compris par les ordinateurs.

Certes, il serait bien plus facile de rédiger un programme en français ordinaire, tel que:

1. Afficher ***
2. Attendre une seconde
3. Afficher *** sur la ligne suivante

Mais, pour que l'ordinateur puisse comprendre une formulation de ce genre, il devrait faire appel à un logiciel bien plus complexe que le BASIC.

Le BASIC est un langage-machine, assez semblable au langage humain et il est relativement facile à maîtriser. Mais comme les ordinateurs excellent bien plus dans la manipulation des valeurs numériques que dans la compréhension de mots français comme "Attendre une seconde", le BASIC demande à l'ordinateur d'effectuer une opération qui change la valeur d'une variable comme

```
FOR T=1 TO 480:NEXT T
```

et d'arriver aux mêmes résultats que s'il comprenait le français comme nous.

Des éléments comme les variables et les numéros de ligne remplissent un rôle essentiel dans la rédaction de programmes en BASIC. Aussi est-il important de se familiariser avec leur utilisation pour réaliser des progrès dans l'étude de la programmation dans ce langage.

LECTURE DES DONNEES

● Préparation des données dans le programme pour les valeurs de variable— READ—DATA

UNE AUTRE METHODE D'AFFECTATION DES VALEURS AUX VARIABLES READ-DATA

Comme nous l'avons constaté, l'emploi judicieux de variables est un des secrets d'une progammation réussie en BASIC. Toutes sortes de valeurs peuvent être données à des variables, qui sont ensuite utilisées pour effectuer des calculs et prendre des décisions. Donner une valeur à une variable est appelé **affectation** d'une valeur.
Passons en revue les méthodes utilisables pour affecter des valeurs à des variables.

Tout d'abord, nous pouvons affecter des valeurs à l'aide de l'énoncé LET.
Pour affecter la valeur 100 à la variable A, nous écrivons:
LET A = 100 (ou simplement A = 100).

Ensuite, nous avons étudié la méthode, utilisant l'énoncé INPUT.
A la différence de l'énoncé LET qui affecte une valeur fixe à une variable du programme, l'énoncé INPUT autorise l'entrée de n'importe quelle valeur via le clavier, pendant l'exécution du programme.
Si, par le clavier, nous entrons 500 quand

```
INPUT A
```

est exécuté et qu'un point d'interrogation (?) est affiché, la valeur 500 sera affectée comme la valeur de A.

L'énoncé READ-DATA est donc un autre moyen du BASIC, utilisable pour affecter des valeurs aux variables.

Essayons ce nouvel élément dans un programme simplifié.

```
10 READ A
20 PRINT A
30 DATA 185
RUN
 185
Ok
```

Quand la ligne 10

```
READ A
```

est exécutée, la valeur donnée dans l'énoncé DATA (185, à la ligne 30 de notre programme) est affectée à la variable A. (La valeur est affichée par l'énoncé PRINT à la ligne 20.) La même chose se vérifie aussi pour les variables alphanumériques.

```
10 READ B$
20 PRINT B$
30 DATA SONY
RUN
SONY
Ok
```

SONY est une valeur à chaîne de caractères, mais il n'est pas nécessaire de l'insérer entre guillemets dans un énoncé DATA.
(Toutefois, si l'on désire inclure une virgule, ou si l'on veut laisser un espace devant un mot, toute la chaîne de caractères devra être comprise entre des guillemets " ".)

```
10 READ B$
20 PRINT B$
30 DATA "SONY, HITBIT"
RUN
SONY, HITBIT
Ok
```

Augmentation du volume des données
Dans les programmes ci-dessus, une seule valeur a été affectée à une variable. Essayons de faire passer à trois ces données (ces valeurs). Pour augmenter jusqu'à trois nos données, nous devrons aussi faire appel à trois variables.

```
10 READ A,B,C
20 PRINT A;B;C
30 DATA 10,20,30
RUN
 10  20  30
Ok
```

Nous utiliserons une virgule pour séparer chaque variable, incluse dans l'énoncé READ. La donnée dans l'énoncé DATA, qui est assignée à chaque variable, est listée dans le même ordre et séparée également par des virgules.

La chose essentielle à ne pas oublier est qu'il est nécessaire de prévoir au moins autant d'énoncés DATA que le demandent les énoncés READ.

SAUVEGARDE DES PROGRAMMES SUR BANDE

- Connexion d'un magnétophone
- Sauvegarde du programme machine sur bande —CSAVE
- Confirmation de la sauvegarde du programme—CLOAD?
- Chargement d'un programme depuis la bande—CLOAD

Quiconque a vu de près ou de loin un grand ordinateur en service aura probablement remarqué, à ses côtés, une autre machine, ressemblant à un magnétophone géant et munie de bandes, tournant à plus ou moins grande vitesse à l'intérieur. En fait, il s'agit bel et bien d'un magnétophone, mais un de première classe. Cette machine a, en effet, pour but de mémoriser les programmes, utilisés par l'ordinateur proprement dit.

Quant à votre ordinateur, sa mémoire est plutôt courte car il ne se souvient d'un programme qu'aussi longtemps qu'il est allumé. Si vous coupez son alimentation, si vous appuyez sur la touche RESET, ou si vous utilisez l'instruction NEW, votre programme disparaît à l'instant de sa mémoire.

Pour pallier cette lacune, il faudrait faire appel à un dispositif d'enregistrement sur lequel vous pourriez ranger (**sauvegarder**) votre programme avant de mettre votre ordinateur à l'arrêt. Ensuite, bien qu'il n'existe plus dans la mémoire de l'ordinateur proprement dit, vous pourriez toujours relire (**charger**) votre programme dans l'ordinateur en le rappelant de la bande dans sa mémoire à la prochaine utilisation.

Heureusement, point n'est besoin de vous équiper d'un périphérique géant comme ceux qui accompagnent les gros ordinateurs. Un simple magnétocassette, que vous possédez sans doute déjà, fera amplement l'affaire.

> Cette section explique comment préserver un programme existant dans la mémoire en le sauvegardant sur cassette. Si vous disposez d'un ordinateur doté d'une unité de disque, vous pouvez ignorer cette partie et passer à la suivante "Sauvegarde des programmes sur disque", où l'on trouvera les explications adéquates.

Si votre magnétocassette n'est pas encore raccordé à l'ordinateur, coupez l'alimentation de ce dernier et consultez les explications suivantes, relatives au branchement. Si vous mettez l'ordinateur hors tension, votre programme sera effacé de la mémoire et vous devrez, bien entendu, l'entrer à nouveau après avoir terminé les connexions.

Connexion d'un magnétophone

Raccordez votre magnétocassette sur le connecteur TAPE de l'ordinateur, en faisant appel au cordon de raccordement du magnétophone. Sur celui-ci existe aussi une prise MIC, destinée à l'enregistrement du son. En outre, vous y trouverez une autre prise pour écouteur, identifiée par un des labels suivants:

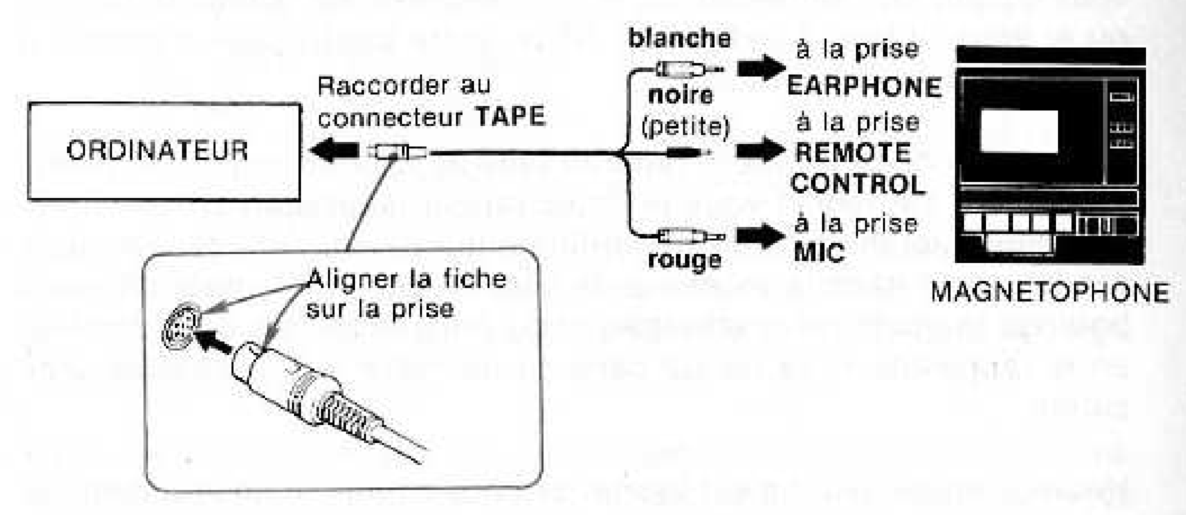
EAR, EARPHONE, MONITOR, MON, ou ⑨

Est aussi prévue parfois une prise

REMOTE

pour la commande à distance.

Brancher le cordon de raccordement de l'ordinateur sur ces prises, comme illustré sur le schéma ci-après.



Si le magnétocassette utilisé n'est pas pourvu d'une prise de télécommande, la fiche noire peut rester débranchée.

SAUVEGARDE DU PROGRAMME MACHINE SUR BANDE CSAVE

L'instruction du BASIC pour sauvegarder un programme informatique sur une bande est CSAVE.

CSAVE "nom de fichier"

L'instruction CSAVE enregistre sur une cassette, en langage intermédiaire, un programme portant un nom de fichier spécifié.

Un nom de fichier est celui que l'on attribue à un programme donné, afin de le distinguer des autres.

Il existe trois règles à respecter pour attribuer un nom de fichier à un programme:

- le nom ne peut pas comporter plus de 6 caractères;
- tout nombre, signe ou lettre de l'alphabet est utilisable;
- l'ordinateur effectue une distinction entre les majuscules et les minuscules.

Donnons donc le nom HEART au "programme cœur" à sauvegarder et sauvegardons-le sur la bande. Tapez l'instruction suivante.

Mais **n'appuyez pas** encore sur la touche [↵].

```
CSAVE "HEART"
```

Ensuite, avant d'appuyer sur la touche [↵], placez le magnétocassette en mode d'enregistrement. Si une commande à distance a été branchée, la bande ne défilera pas. L'enregistrement commencera seulement au moment où vous actionnez la touche [↵]. Le programme passe alors par le connecteur TAPE de l'ordinateur et il s'enregistre sur la bande magnétique.

Si une télécommande n'est pas raccordée, la bande commencera à défiler dès l'instant où le magnétocassette est placé en mode d'enregistrement. Confirmez que la bande défile et appuyez immédiatement sur la touche [↵].

CONFIRMATION DE LA SAUVEGARDE DU PROGRAMME CLOAD?

Après l'avoir enregistré sur une bande, veuillez toujours prendre la précaution de vérifier que le programme y est correctement sauvegardé. A cet effet, rebobinez la bande jusqu'au point précédant le début de l'enregistrement du programme en question. (Sur certains magnétocassettes, la prise de télécommande devra être débranchée pour permettre le rebobinage de la bande.) Ajustez ensuite le réglage du volume du magnétophone à sa position moyenne et entrez alors l'instruction suivante.

```
CLOAD? "HEART"
```

CLOAD? "nom de fichier"

L'instruction CLOAD? vérifie que le programme, enregistré sur la bande, et celui qui reste dans la mémoire de l'ordinateur sont rigoureusement identiques.

Après avoir tapé l'instruction CLOAD?, appuyez sur la touche [↵], puis placez le magnétocassette en mode de lecture **(PLAY)**. (Si la prise de télécommande est branchée et que vous appuyez sur PLAY avant d'actionner la touche [↵], la bande ne défilera pas avant d'appuyer sur la touche [↵].) A l'instant où la bande arrive au début de l'enregistrement du programme,

```
Found:HEART
```

sera affiché. Ce message vous signale que l'ordinateur a retrouvé, sur la bande, le fichier appelé HEART. Il effectue alors une comparaison de ce programme avec celui qui subsiste dans sa mémoire. Le temps requis pour cette comparaison dépend de la longueur du programme. Si les deux programmes sont identiques,

```
OK
```

sera affiché à l'écran.

Si le programme, mémorisé par l'ordinateur, ne correspond pas à celui de la bande, le message

```
Verify error
```

sera affiché. Ce message "Verify error" signifie que le programme n'a pas été correctement enregistré sur la bande. Dans ce cas, il faudra procéder à une nouvelle sauvegarde, en faisant de nouveau appel à l'instruction CSAVE.

Il peut arriver parfois que le message

```
Found:HEART
```

ne soit pas affiché, bien que la bande ait défilé au-delà du début de l'enregistrement du programme recherché. Ceci provient habituellement d'un réglage trop élevé ou insuffisant de la commande du volume du magnétophone. Vérifiez le niveau du volume et recommencer l'instruction CLOAD?. Lorsque le message "Found: HEART" est affiché, observez bien le réglage de la commande du volume sur le magnétocassette et souvenez-vous en, car c'est le niveau correct à utiliser avec l'instruction CLOAD?.

CHARGEMENT D'UN PROGRAMME DEPUIS LA BANDE CLOAD

Pour faire lire (**charger**) le programme de la cassette dans la mémoire de l'ordinateur, vous devrez utiliser l'**instruction CLOAD.**

CLOAD "nom de fichier"

L'instruction CLOAD charge dans la mémoire de l'ordinateur, le programme, dont le nom de fichier a été spécifié.

Vous devez écrire, dans l'espace "nom de fichier" de l'énoncé, le nom du fichier que vous souhaitez transférer de la bande vers l'ordinateur. Dans notre exemple précédent, ce devrait être:

```
CLOAD "HEART"
```

Appuyez sur la touche [↵] et placer le magnétocassette en mode de lecture (PLAY). (Si la prise de télécommande est branchée et que vous appuyez sur PLAY avant d'actionner la touche [↵], la bande ne commencera à défiler qu'après avoir appuyé sur la touche [↵].) Lorsque la bande arrive à l'endroit où le programme est enregistré,

```
Found:HEART
```

est affiché. Après que le programme a été complètement chargé dans la mémoire de l'ordinateur, l'écran se présente comme suit:

```
CLOAD "HEART"
Found:HEART
Ok
```

Si, au lieu de Ok, apparaît le message suivant,

```
Device I/O error
```

c'est que le programme n'a pas été correctement chargé dans la mémoire de l'ordinateur. Modifiez alors le niveau du volume du magnétocassette et utilisez à nouveau l'instruction CLOAD pour charger à nouveau le programme.

Une fois que le programme a été chargé, entrez l'instruction LIST pour confirmer que le programme se trouve bien dans la mémoire de l'ordinateur, puis déroulez ce programme par l'instruction RUN.

SAUVEGARDE DES PROGRAMMES SUR DISQUE

- Mise en forme d'un disque—CALL FORMAT
- Sauvegarde du programme machine sur disque—SAVE
- Confirmation de la sauvegarde du programme—FILES
- Chargement d'un programme depuis le disque—LOAD
- Effacement d'un programme sur disque—KILL

Cette partie concerne les ordinateurs, accompagnés d'une ou de plusieurs unités de disque. Si votre ordinateur n'en est pas équipé, la section précédente "Sauvegarde des programmes sur bande" vous aura fourni les informations nécessaires à la sauvegarde de vos programmes sur cassette.
Si votre ordinateur MSX2 n'est pas équipé d'une unité de disque, rien ne vous empêche pour autant de sauvegarder vos programmes sur disquettes, en faisant appel à une unité de disque souple, telle que la SONY HBD-50. Dans ce cas, il est nécessaire de raccorder l'unité de disque en question à l'ordinateur et de lancer le MSX-Disk BASIC. Veuillez consulter les explications sous "Utilisation d'une unité de disque externe" ci-après.

Utilisation d'une unité de disque externe

(1) Couper l'alimentation de l'ordinateur et de l'unité de disque extérieure.
(2) Brancher la cartouche d'interface d'unité de disque dans la fente adhoc sur l'ordinateur.
(3) Mettre l'unité de disque sous tension. Ensuite, mettre le moniteur de télévision et l'ordinateur sous tension. Après une brève pause, l'affichage suivant apparaît sur l'écran.

```
Enter date (Y-M-D):
```

(4) Si le message ci-dessus apparaît, appuyer sur la touche [↵].

Telles sont les démarches nécessaires à la mise en service du MSX-Disk BASIC.

FORMATAGE D'UN DISQUE CALL FORMAT

Le simple fait de couper l'alimentation de votre ordinateur peut suffire à ruiner le programme pour la rédaction duquel vous avez durement travaillé. C'est pourquoi, avant de mettre votre ordinateur au repos, vous devrez sauvegarder votre programme sur une disquette, installée dans l'unité de disque. Si vous prenez cette précaution élémentaire, vous aurez toujours la possibilité, par la suite, de rappeler le programme, de la disquette vers la mémoire de l'ordinateur, et de l'utiliser à nouveau.

Cependant, avant de pouvoir être utilisé, un nouveau disque souple doit être **formaté**.

Par cette opération de formatage, l'ordinateur écrit sur la disquette des informations spéciales, suivant une série de règles fixes, qui remplissent le rôle de guides, destinés à permettre à la machine de localiser immédiatement l'emplacement de chaque fichier sur le disque. On pourrait comparer cette opération au tracé de lignes sur une feuille de papier vierge, afin de savoir où l'on doit écrire et quels espaces laisser entre les lignes. Heureusement, vous n'avez pas à vous préoccuper du processus de formatage d'un disque; l'ordinateur s'en charge. Qu'il vous suffise de vous rappeler que **tout disque doit être formaté avant d'étré utilisé.**

Remarque: Le fait de formater un disque a pour conséquence d'effacer toutes les informations qui s'y trouvaient déjà.

Procédons maintenant à la mise en forme d'un nouveau disque.

(1) Exécuter l'instruction CALL FORMAT.

Quand vous appuyez sur la touche [↵],

```
Drive name ? (A,B)
```

apparaît.

Ce message demande si vous désirez formater le disque dans l'unité A ou dans l'unité B. Si vous n'utilisez qu'une seule unité, ce sera toujours la "A" et, par conséquent, vous devez appuyer sur la touche [A].

(2) Si votre unité de disque accepte des disques double face,

```
1 - Single sided, 9 sectors
2 - Double sided, 9 sectors
```

sera affiché ensuite. Si l'unité est du type à simple face, appuyez sur la touche [1]; si elle est du type à double face, appuyez sur la touche [2].

Quand vous avez actionné la touche,

```
Strike a key when ready
```

apparaît. (Avec une unité de disque simple face, telle que la SONY HBD-50, ce message apparaît quand vous actionnez la touche [A].)

(3) Installez un nouveau disque dans l'unité et appuyez sur une touche quelconque du clavier. L'ordinateur se charge alors de formater le disque. Lorsque l'opération est achevée, l'écran affiche:

```
Format complete
Ok
```

SAUVEGARDE DU PROGRAMME MACHINE SUR DISQUE SAVE

L'instruction SAVE s'emploie pour sauvegarder sur un disque, en langage intermédiaire, un programme qui se trouve dans la mémoire de l'ordinateur.

SAVE "A: nom fichier. nom type"

La lettre A: spécifie l'unité de disque A, mais si l'on n'utilise qu'une seule unité, cet élément peut être omis.

Le **nom de fichier** est celui qui a été attribué à un programme, afin de le distinguer des autres.

Quatre règles doivent être respectées dans l'attribution d'un nom de fichier à un programme:

- Le nom ne peut pas dépasser 8 caractères.
- Tout nombre, signe ou lettre de l'alphabet est utilisable.
- L'ordinateur ne fait pas de distinction entre les majuscules et les minuscules.
- Les caractères suivants ne sont pas utilisables comme élément d'un nom de fichier:
  , . ; : " * ? et espace

Le nom de type a pour but d'indiquer le type de fichier. Il se compose d'un point (.), suivi de trois lettres au maximum. Comme le type de fichier que nous allons sauvegarder est un fichier BASIC, nous pourrions utiliser ".BAS" comme extension, afin d'indiquer que le fichier comporte un programme, rédigé en langage BASIC. Ce nom pourrait être omis, mais à mesure que vous sauvegarderez d'autres types de fichier sur vos disques, vous comprendrez qu'il est pratique d'inclure toujours cette référence, comme partie de l'instruction SAVE. Dès le début, prenez donc l'habitude d'ajouter le nom de type .BAS au nom de tous vos programmes, écrits en BASIC.

A présent, sauvegardons notre "programme cœur" sur le disque en lui donnant le nom de fichier HEART et l'extension .BAS.

Entrons l'instruction suivante:

```
SAVE "HEART.BAS"
```

Une poussée sur la touche ↵ mettra en marche l'unité de disquette. Une fois que le programme est sauvegardé, le message Ok est affiché.

CONFIRMATION DE LA SAUVEGARDE DU PROGRAMME FILES

L'instruction FILES s'emploie pour vérifier si le programme a bien été sauvegardé sur la disquette.

FILES

L'instruction FILES affiche sur l'écran les noms de fichier et les noms de type de tous les fichiers, sauvegardés sur un disque.

```
FILES
HEART    .BAS
Ok
```

Si vous exécutez l'instruction FILES, les lettres HEART.BAS, c'est-à-dire le nom complet du fichier qui vient d'être sauvegardé, devrait être affiché. Si d'autres fichiers se trouvent sur le disque, leur nom apparaîtra également.

CHARGEMENT D'UN PROGRAMME DEPUIS LE DISQUE LOAD

Avant de charger un programme sur le disque, utilisez l'instruction NEW pour vider la mémoire de l'ordinateur.

LOAD est l'instruction utilisée pour rappeler, dans la mémoire de l'ordinateur, un programme se trouvant sur le disque.

LOAD "A: nom fichier .nom type"

La lettre A: peut être omise. Le nom de fichier et son type doivent être identiques à ceux qui ont servis lors de la sauvegarde du programme à rappeler. Par conséquent, on entrera:

```
LOAD "HEART.BAS"
```

Une pression sur la touche ↵ mettra en service l'unité de disque qui chargera le programme. Lorsque cette opération est achevée, le message Ok est affiché. Par une instruction LIST, confirmez que le programme en question a bien été placé dans la mémoire de l'ordinateur et, par l'instruction RUN, exécutez-le.

Chaque fois que vous souhaiterez utiliser un des programmes, sauvegardés sur un disque, vous pourrez ainsi le charger dans la mémoire de l'ordinateur par l'instruction LOAD.

EFFACEMENT D'UN PROGRAMME DU DISQUE KILL

La sauvegarde de nombreux programmes sur un disque entraîne progressivement sa saturation.

L'instruction KILL vous permettra d'éliminer du disque les programmes dont vous n'avez plus besoin.

KILL "A: nom fichier .nom type"

La lettre A: peut être omise. Pour effacer le programme HEART .BAS qui avait été sauvegardé sur le disque, entrez l'instruction:

```
KILL "HEART.BAS"
```

Chapitre 3
Variables en tableau

UN PROGRAMME UTILISANT DES VARIABLES EN TABLEAU

- La déclaration de variable en tableau—DIM

POUR UTILISER LES VARIABLES EN TABLEAU DIM

Utilisons quelques variables en tableau dans un programme réel. Nous ferons appel à deux variables en tableau de type chaîne, N$(N) et T$(N), et nous leur ferons contenir chacune cinq informations différentes. La valeur de N ira donc de 0 à 4.
Nous utiliserons la variable en tableau N$(N) pour le nom de personnes et la variable en tableau T$(N) pour leur numéro de téléphone. Elles renfermeront ainsi les données suivantes.

Variable en tableau	Donnée	Variable en tableau	Donnée
N$(0)	PETER	T$(0)	111-2222
N$(1)	PAUL	T$(1)	222-3333
N$(2)	MARY	T$(2)	333-4444
N$(3)	TOM	T$(3)	444-5555
N$(4)	SUSIE	T$(4)	555-6666

Affectation des données aux variables en tableau
Comme nous l'avons fait pour les variables ordinaires, utilisons l'énoncé LET, l'énoncé INPUT ou l'énoncé READ—DATA pour affecter les données aux variables en tableau. Dans le cas qui nous occupe, nous utilisons un énoncé READ—DATA.

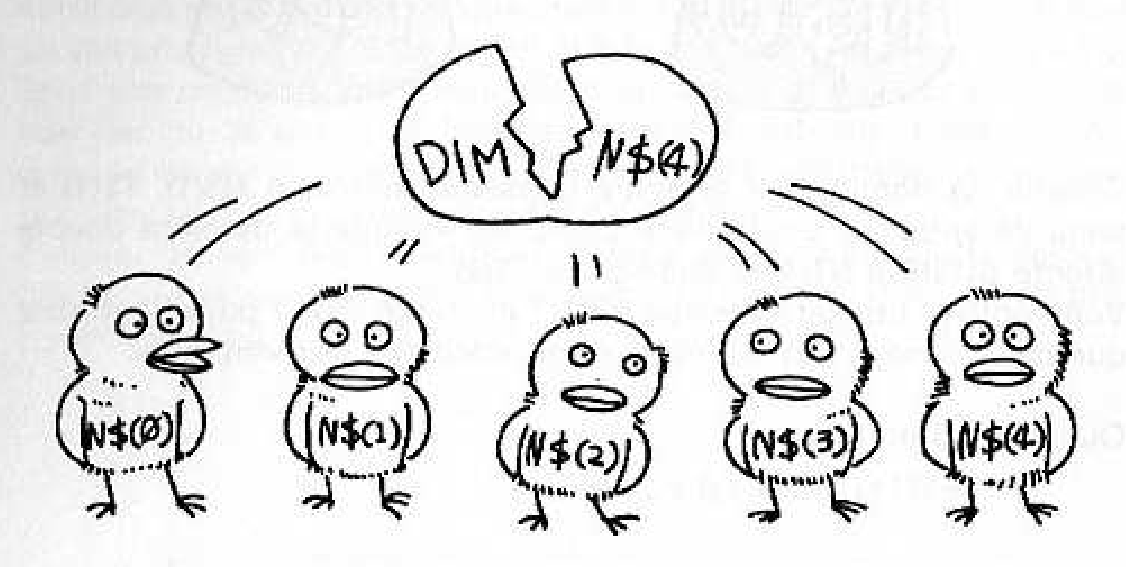
```
10 DIM N$(4),T$(4)
20 FOR L=0 TO 4
30 READ N$(L),T$(L)
40 NEXT L
100 END
110 DATA PETER,111-2222
120 DATA PAUL,222-3333
130 DATA MARY,333-4444
140 DATA TOM,444-5555
150 DATA SUSIE,555-6666
```

Nous avons utilisé un tout nouvel énoncé à la ligne 10.

```
10 DIM N$(4),T$(4)
```

L'énoncé DIM s'emploie pour déclarer combien de variables en tableau seront utilisées dans un programme et combien de variables elles contiendront.

DIM N$(4), T$(4) déclare que N$, qui aura 5 variables de N$(0) à N$(4), et T$, avec 5 variables de T$(0) à T$(4) seront utilisées dans le programme.

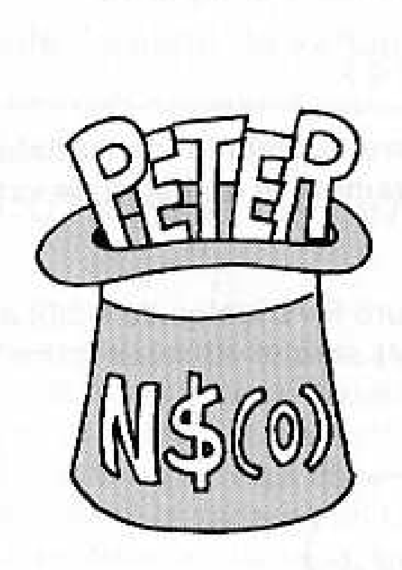


DIM nom de variable en tableau (valeur maximum)

L'énoncé DIM spécifie la valeur maximale du nombre de variables entre les parenthèses ( ) d'une variable en tableau. Si la valeur maximale est N, la variable en tableau aura des variables de 0 à N, ce qui sera égal à la valeur de N plus 1.

Souvenez-vous toujours de déclarer une variable en tableau avec un énoncé DIM avant d'utiliser la variable en tableau dans un programme.

Après que les variables en tableau ont été déclarées à la ligne 10, la boucle FOR—NEXT de lignes 20 à 40 affecte les données aux variables en tableau. A la première exécution de la boucle, la valeur de L est 0; aussi l'énoncé READ de la ligne 30 affecte-t-il PETER en N$(0) et 111-2222 en T$(0).





Ensuite, la donnée est affectée consécutivement à N$(1), T$(1) et ainsi de suite. Le programme prend fin lorsque la dernière boucle affecte SUSIE à N$(4) et 555-6666 à T$(4).
Vous pouvez utiliser l'énoncé PRINT en mode direct pour confirmer que ces données ont bel et bien été affectées aux variables.

Quand vous entrerez

```
PRINT N$(0),T$(0)
```

```
PRINT N$(0),T$(0)
PETER          111-2222
OK
```

sera affiché. Si vous entrez PRINT N$(1), PAUL sera affiché, ou si vous entrez PRINT T$(2), vous obtiendrez le numéro de téléphone 333-4444.

Nous voici arrivés à la fin de notre introduction au MSX2-BASIC. Au cours de ce cheminement, vous avez appris les rudiments indispensables à la rédaction de programmes en langage BASIC.
Désormais, vous pourrez utiliser ces instructions pour écrire vos propres programmes. Nous tenons à vous y encourager car le meilleur moyen de progresser est, tout d'abord, d'apporter certaines modifications aux programmes rencontrés dans ce manuel. Nos explications ne sont certes pas exhaustives mais, avec un peu de réflexion, vous parviendrez sûrement à rendre ces programmes plus faciles et plus adaptés à vos besoins.
En terminant, une dernière remarque. Ne craignez pas d'endommager votre ordinateur en laissant une erreur se glisser dans la formulation de vos programmes. Si ce devait être le cas, votre machine la lirait et vous transmettrait tout simplement un message d'erreur pour vous signaler où se trouve la "bogue" (erreur) éventuelle. Dans ce cas, vous le savez à présent, l'instruction LIST afficherait votre programme et vous pourriez localiser la difficulté.
Essayer de nouvelles choses et corriger ses erreurs est le meilleur moyen de progresser dans la découverte des richesses de la programmation en BASIC.

ETAPES AVANCEES

Chapitre 4
La fonction de commutation de mémoire

L'ENONCE SET

- La fonction de commutation de mémoire
- Addition d'un titre—SET TITLE
- Changement de l'énoncé de signalisation—SET PROMPT
- Spécification du mot de passe—SET PASSWORD
- Changement de la position de l'affichage sur l'écran—SET ADJUST
- Réglage de la tonalité 'bip'—SET BEEP
- Spécification de l'état initial sur l'écran—SET SCREEN

LA FONCTION DE COMMUTATION DE MEMOIRE

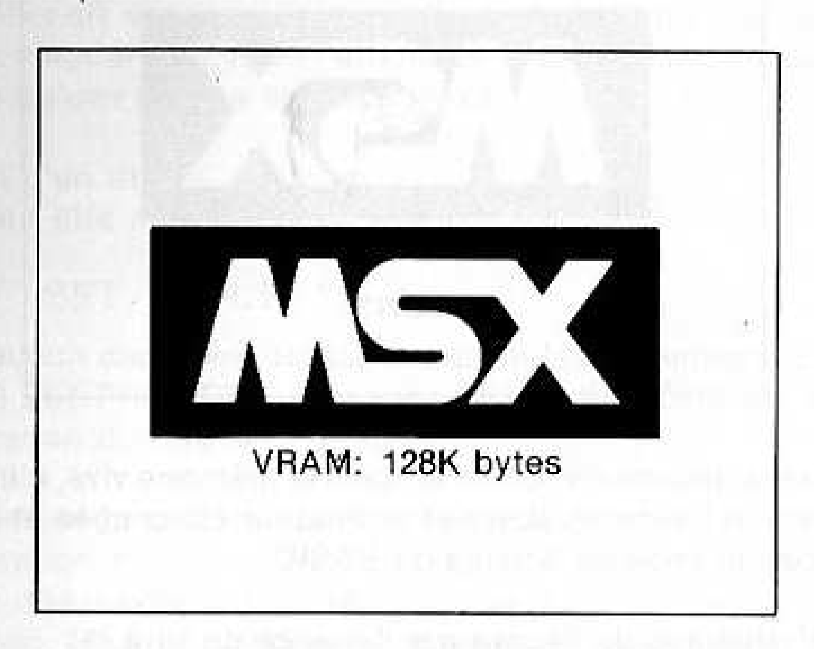
Le langage MSX2-BASIC dispose d'une fonction spéciale, appelée fonction de commutation de mémoire. Elle permet de changer les réglages initiaux de l'ordinateur lors de la mise en marche du BASIC et de faire conserver les réglages modifiés par une mémoire vive (RAM), alimentée par pile, dans le CI du rythmeur.

La mémoire vive (RAM) constitue la mémoire principale de l'ordinateur. (Dans la section ETAPES ELEMENTAIRES, nous avions parlé à son sujet de "mémoire de l'ordinateur".) Le fait de mettre l'ordinateur hors tension a pour résultat d'effacer le contenu de la mémoire vive, mais une mémoire vive, alimentée par pile, aura l'avantage de conserver son contenu, même quand l'alimentation sera coupée. Par conséquent, si nous parvenons à faire conserver les réglages initiaux du BASIC par cette mémoire vive spéciale, ces réglages seront encore utilisables lorsque l'ordinateur sera remis ultérieurement sous tension.

Les réglages initiaux suivants peuvent être changés et conservés dans la mémoire vive, alimentée par pile.
- Les titres et énoncés de signalisation
- Le mot de passe
- La position de l'affichage sur l'écran
- Le type et le volume de la tonalité 'bip'
- Les réglages de l'énoncé SCREEN

ADDITION D'UN TITRE SET TITLE

A la mise en service du BASIC, l'affichage suivant est obtenu sur l'écran avant que ne soit affiché Ok.



Vous avez le loisir d'ajouter le titre de votre choix en dessous de la ligne "VRAM: 128K bytes" (ou VRAM 64K bytes) de cet affichage. Ce titre est défini par l'énoncé SET TITLE.

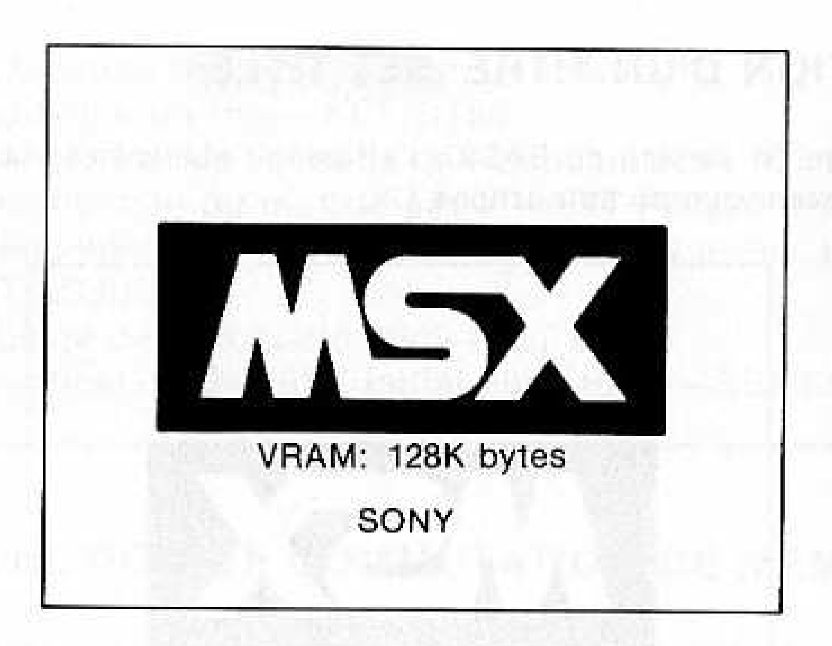
SET TITLE ["énoncé de titre"], [couleur]

Le titre peut contenir jusqu'à six caractères. De plus, quatre combinaisons différentes de couleurs peuvent être spécifiées pour l'affichage du logo MSX. Si vous le souhaitez, vous pouvez omettre le titre et spécifier uniquement la combinaison de couleur de votre choix.

Pour afficher le titre "SONY", exécutez l'énoncé SET TITLE suivant en mode direct.

```
SET TITLE "SONY"
```

Après avoir exécuté cet énoncé, appuyez sur la touche RESET ou coupez brièvement l'ordinateur et remettez-le sous tension. Le titre apparaîtra alors comme illustré ci-dessous.

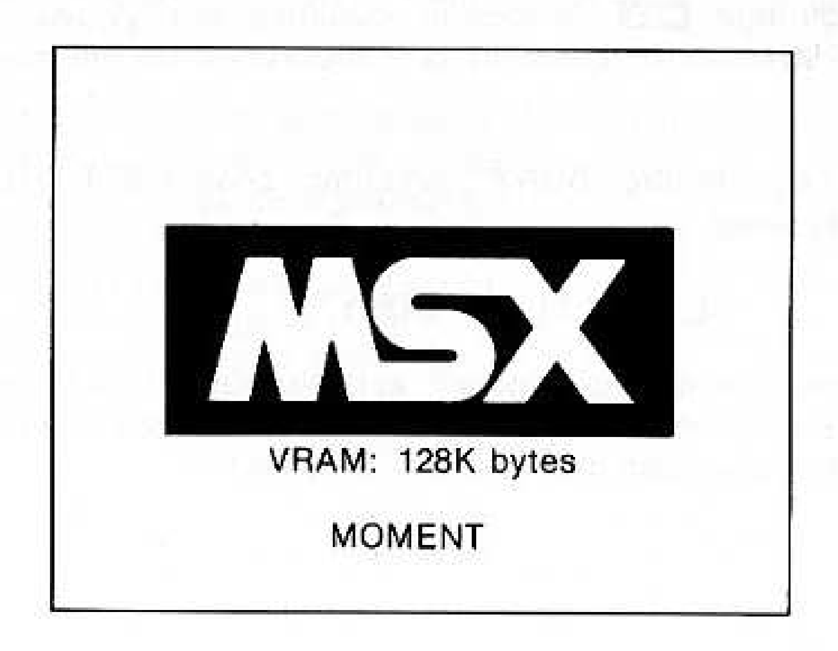


Ce titre sera désormais conservé dans la mémoire vive, alimentée par pile, même si l'alimentation de l'ordinateur est coupée et il sera affiché à chaque mise en service du BASIC.

Gel de l'affichage de l'écran par l'énoncé du titre
Si vous spécifiez un titre comportant exactement six caractères, l'affichage sera "gelé" sur l'écran à l'apparition du titre. Par exemple, si vous spécifiez

```
SET TITLE "MOMENT"
```

l'écran présentera l'affichage suivant lors de la mise en service ultérieure du BASIC.



Comme "MOMENT" comporte exactement six caractères, l'affichage subsistera sur l'écran jusqu'à ce qu'une touche du clavier soit actionnée. Le message Ok sera ensuite affiché et le BASIC sera prêt à l'utilisation. Si, au lieu des caractères, vous utilisez l'énoncé SET TITLE pour spécifier six espaces vierges à l'aide de la barre d'espacement, aucun titre n'apparaîtra, mais l'affichage **MSX** subsistera sur l'écran jusqu'à ce qu'une touche soit actionnée.

Annulation d'un titre
Un titre peut être effacé en exécutant:

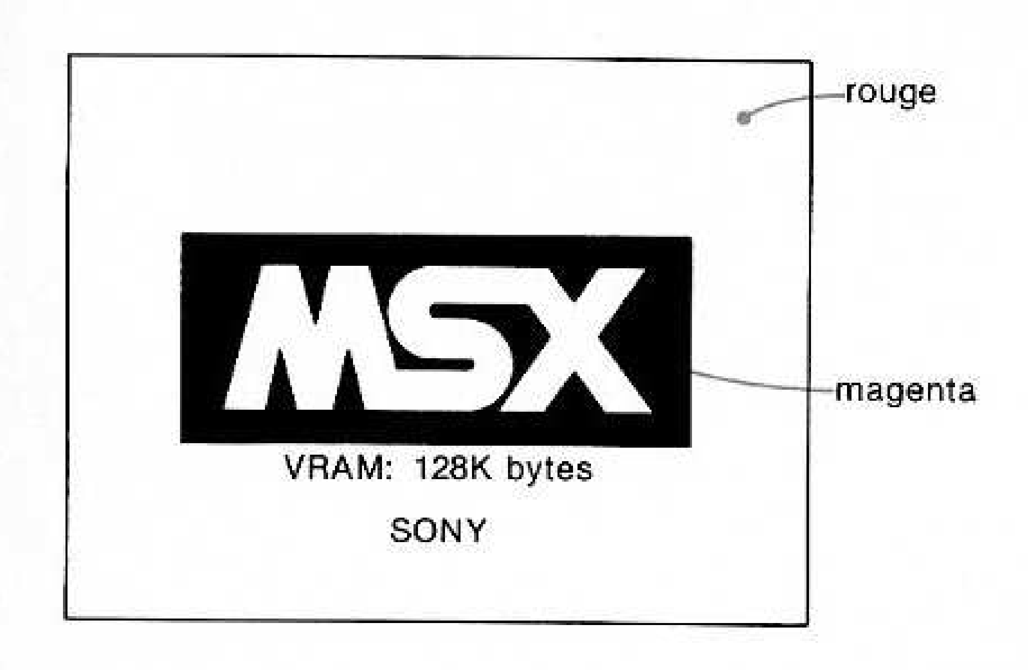
```
SET TITLE ""
```

Ne tapez aucun caractère ou espace entre les guillemets. En outre, si l'énoncé SET PROMPT ou l'énoncé SET PASSWORD est exécuté, la spécification du titre sera annulée.

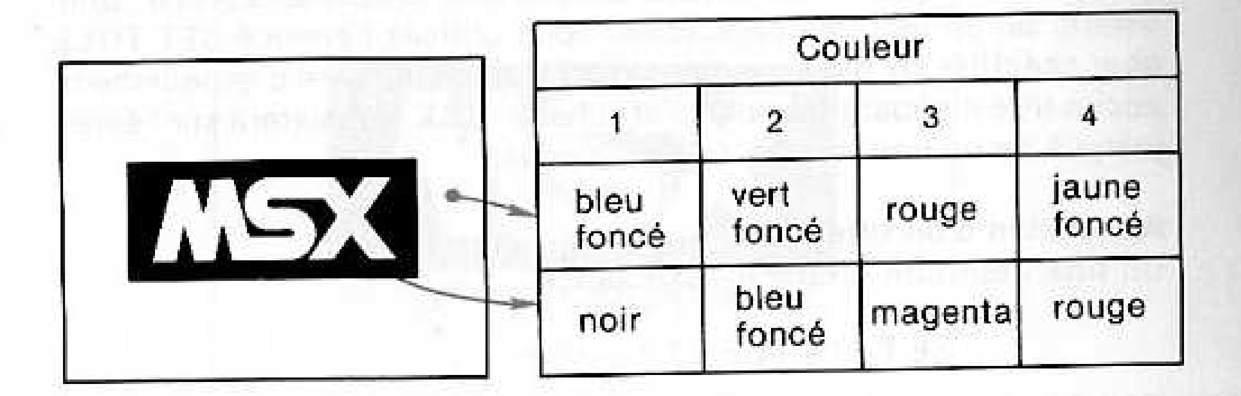
Changement de la couleur de l'affichage du titre
La spécification couleur dans l'énoncé SET TITLE peut servir pour définir une des quatre différentes combinaisons couleurs de l'écran quand le logo **MSX** est affiché. La couleur est spécifiée par des chiffres de 1 à 4. Quand

```
SET TITLE "SONY",3
```

est exécuté, l'écran présente les couleurs suivantes.



Le tableau suivant indique les combinaisons de couleurs qui peuvent être spécifiées.



Couleur			
1	2	3	4
bleu foncé	vert foncé	rouge	jaune foncé
noir	bleu foncé	magenta	rouge

CHANGEMENT DE L'ENONCE DE SIGNALISATION SET PROMPT

Le message Ok apparaît quand le BASIC est prêt pour recevoir vos instructions. "Ok" est ce que nous appellerons ici l'énoncé de signalisation. **Ce message peut être changé en tout mot comportant six caractères au maximum.** Par exemple, l'énoncé SET PROMPT ci-après changera le message de "Ok" en "Please".

```
SET PROMPT "Please"
```

Une fois que le message a été changé par l'énoncé SET PROMPT, le nouveau message sera affiché à chaque mise en service du BASIC et il ne pourra plus être changé sans exécuter un autre énoncé PROMPT, SET TITLE ou SET PASSWORD. Quand un énoncé SET TITLE ou un énoncé SET PASSWORD est exécuté, le message PROMPT repassera au message "Ok" initial.

```
SET PROMPT "Please"
Please
█
```

SPECIFICATION D'UN MOT DE PASSE
SET PASSWORD

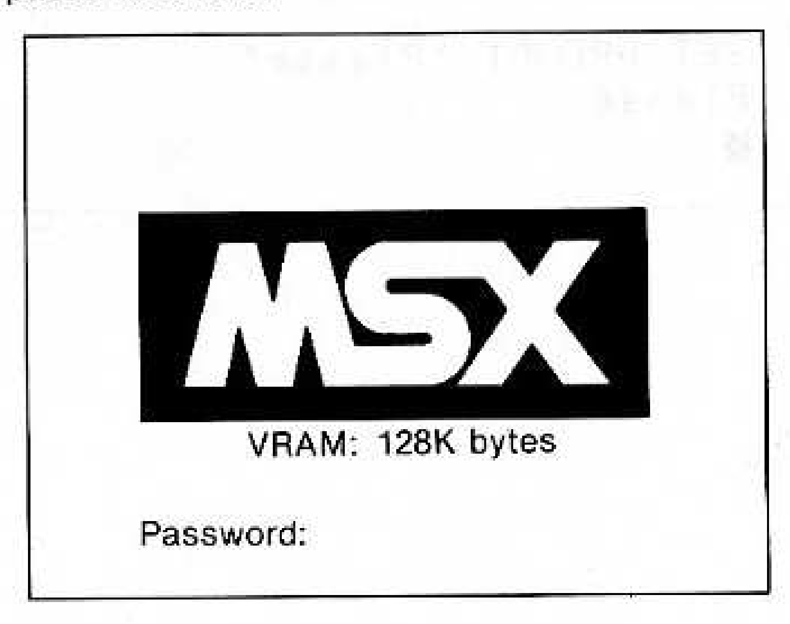
Un mot de passe est un terme, connu de vous uniquement et qui doit être entré avant de pouvoir mettre le BASIC en service. L'énoncé SET PASSWORD est utilisé pour définir ce mot de passe.

SET PASSWORD "mot de passe"

Le mot de passe peut être une chaîne de caractères, comportant jusqu'à 255 caractères. Par exemple, si l'on désire utiliser "I LIKE BASIC" (12 caractères, les espaces compris) comme mot de passe, on devra entrer:

```
SET PASSWORD "I LIKE BASIC"
```

Une fois que cet énoncé a été exécuté et lorsque le BASIC est mis en service, l'écran fera apparaître l'affichage suivant jusqu'à ce que le mot de passe est entré.



Le BASIC ne sera pas en service aussi longtemps que vous n'aurez pas entré le mot de passe I LIKE BASIC et appuyé sur la touche [↵]. Si le mot de passe n'est pas entré ou si l'on entre un mot erroné, le BASIC reste inactif. De cette façon, vous seul (ou tout autre personne connaissant le mot de passe) pourrez utiliser l'ordinateur.

Remarque

Une fois qu'un mot de passe a été défini, son entrée sera requise également avant l'utilisation de logiciels en cartouche, tels que pour les jeux vidéo.

Annulation du mot de passe
Un mot de passe sera effacé par exécution d'un énoncé SET PROMPT ou d'un énoncé SET TITLE. Par exemple quand

```
SET PROMPT "Ok"
```

est exécuté, le message de signalisation redevient Ok et le mot de passe est annulé. Autrement dit, le dernier énoncé SET à exécuter, — que celui-ci soit SET PROMPT, SET TITLE ou SET PASSWORD — devient l'énoncé valide et tout énoncé SET exécuté auparavant est annulé.

En cas d'oubli du mot de passe
Si vous avez oublié le mot de passe défini, maintenez enfoncées les touches GRAPH et STOP et appuyez sur RESET. Maintenez la pression sur ces trois touches jusqu'à apparition du logo **MSX** sur l'écran. Après avoir confirmé que

```
Password:
```

n'est plus affiché sur l'écran, relâchez les trois touches et le BASIC sera mis en service.
Une autre manière d'éviter le mot de passe est d'appuyer et de maintenir enfoncées les touches GRAPH et STOP quand vous allumez l'interrupteur d'alimentation secteur de l'ordinateur.

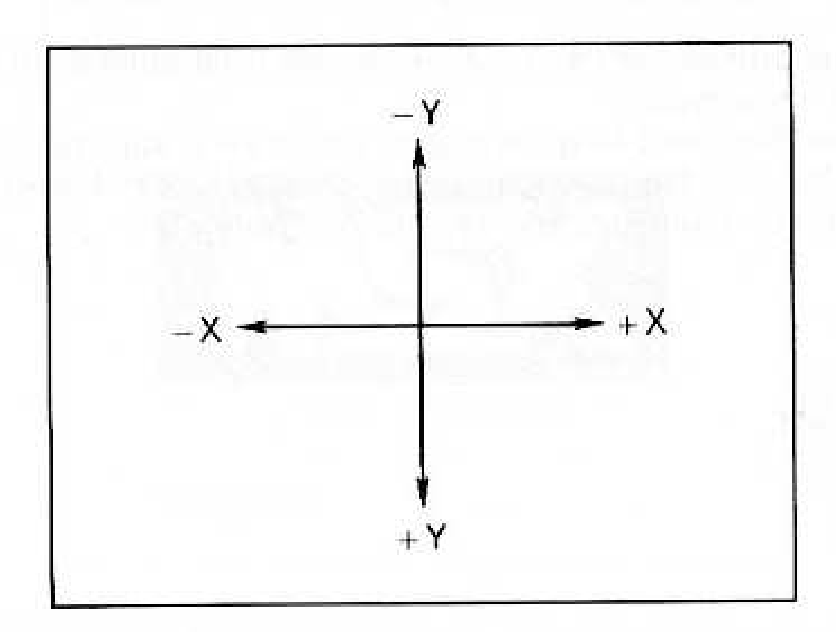
CHANGEMENT DE LA POSITION DE L'AFFICHAGE SUR L'ECRAN SET ADJUST

Il arrive, avec certains types d'écran moniteur, que la zone d'affichage ne soit pas correctement centrée sur l'écran. L'énoncé SET ADJUST permet d'ajuster cette zone d'affichage, de façon qu'elle arrive en plein centre de l'écran utilisé.

SET ADJUST (X,Y)

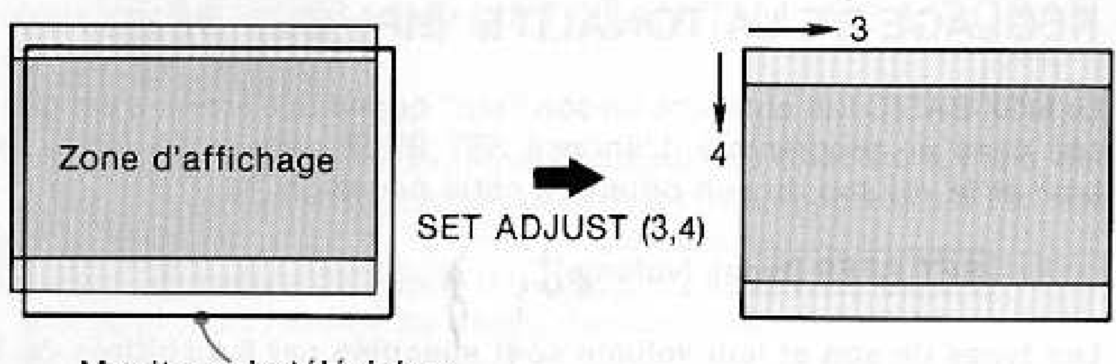
Les chiffres de −7 à +8 peuvent être spécifiés pour les axes X et Y. La position de la zone d'affichage sera changée d'un point en coordonnée par chaque changement de la valeur du chiffre spécifié.

Les réglages implicites initiaux des axes X et Y sont 0. L'affichage se déplacera vers la droite en spécifiant une valeur positive (+) pour X, tandis qu'une valeur négative (−) le déplacera vers la gauche. De même, une valeur positive (+) pour Y déplacera l'affichage vers le bas, tandis qu'une valeur négative (−) le déplacera vers le haut.



Par exemple, si la zone d'affichage est décentrée vers le haut et la gauche comme sur l'illustration ci-dessous, vous pourrez la recentrer en la déplaçant de 3 points de coordonnées vers la droite et de 4 points vers le bas en exécutant

```
SET ADJUST (3,4)
```



Utilisons à présent l'énoncé SET ADJUST dans un petit programme pour obtenir un effet intéressant.

```
10 SCREEN 2
20 CIRCLE (125,95),60
30 FOR Y=-7 TO 8
40 FOR X=-7 TO 8
50 SET ADJUST (X,Y)
60 NEXT X
70 NEXT Y
80 GOTO 30
```

REGLAGE DE LA TONALITE 'BIP' SET BEEP

Le MSX-BASIC fait entendre un son "bip" quand une erreur s'est glissée dans un programme. L'énoncé SET BEEP permet de définir le type et le volume du son obtenu à cette occasion.

SET BEEP [type], [volume]

Les types de son et leur volume sont spécifiés par les chiffres de 1 à 4. Le chiffre 1 fournit le volume le plus réduit et le chiffre 4 le plus fort.

Nº	1	2	3	4
Type	♬	♩	♩ ♩	♬♪
Volume	le plus faible	←	→	le plus fort

Pour régler le type de "bip" à la position 3 et son volume au niveau 4, on devra exécuter:

```
SET BEEP 3,4
```

SPECIFICATION DE L'ETAT INITIAL DE L'ECRAN
SET SCREEN

L'énoncé SCREEN sera expliqué en détails au Chapitre 5, mais voici une simple liste des réglages qui peuvent être apportés par l'énoncé SCREEN.

- mode caractère (SCREEN 0 ou 1)
- mise en/hors service du déclic des touches
- type d'imprimante
- vitesse de transfert cassette
- mode entrelacement

En outre, le nombre de caractères sur une ligne peut être spécifié par l'énoncé WIDTH.
L'énoncé KEY ON/OFF a pour but de spécifier si le contenu des touches de fonction est listé ou non dans le bas de l'écran.
Quand à l'énoncé COLOR, il sert à spécifier la couleur de l'avant-plan, celle du fond et celle du pourtour.

Lorsque l'énoncé SET SCREEN est exécuté en mode direct, les réglages actuels, effectués par l'énoncé SCREEN, l'énoncé WIDTH, l'énoncé COLOR et l'énoncé KEY ON/OFF deviennent les réglages initiaux, en vigueur à la mise en service du BASIC.

SET SCREEN

Par exemple, quand

```
SET SCREEN
```

est exécuté après avoir défini le mode SCREEN 1 et avoir spécifié le noir comme couleur de l'avant-plan, le gris comme couleur du fond et le bleu clair comme couleur du pourtour de la façon suivante:

```
SCREEN 1
COLOR 1,14,5
```

ces couleurs seront les couleurs qui apparaîtront sur l'écran lors de la mise en service ultérieure du BASIC.

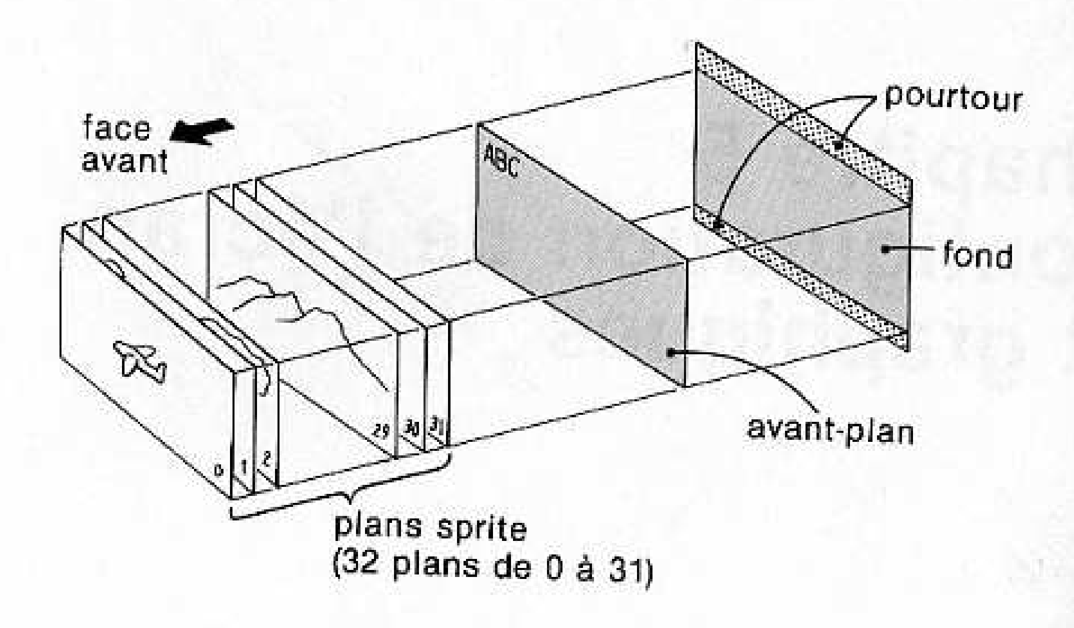
Chapitre 5
Configuration de l'écran et graphiques

MODE ECRAN

- Configuration de l'écran
- Définition du mode—SCREEN
- Spécification du nombre de caractères par ligne—WIDTH
- Coordonnées du mode graphique et spécification STEP

CONFIGURATION DE L'ECRAN

Le schéma suivant illustre la configuration de l'écran en MSX2-BASIC.



DEFINITION DU MODE SCREEN

Les caractères (lettres, chiffres, etc.) sont affichés en **modes caractère**, tandis que les schémas et les dessins le sont par points en **modes graphiques.** En MSX2-BASIC, il existe un total de neuf modes d'écran texte et graphique et l'on fera appel à l'énoncé SCREEN pour spécifier le mode écran à utiliser.

> **SCREEN [numéro de mode], [taille sprite], [déclic touches], [vitesse de transfert], [type imprimante], [entrelacement]**

L'énoncé SCREEN spécifie le mode écran, la taille de sprite, la mise en/hors service du déclic des touches, la vitesse de transfert sur cassette, le type d'imprimante et le mode d'entrelacement.

Neuf modes écran peuvent être choisis par l'énoncé SCREEN. (Les parties de l'énoncé SCREEN, autres que le mode écran, seront expliquées à partir de la page 149.)

Les modes SCREEN

N° mode	Mode	Affichage écran	Couleur	Page	Sprite
0	Texte	Maxi 80 caractères horizontaux, 24 lignes verticales	Fonction palette de couleur, 16 couleurs/512 couleurs	—	Inutilisable
1		Maxi 32 caractères horizontaux, 24 lignes verticales	Fonction palette de couleur, 16 couleurs/512 couleurs	—	Utilisé
2	Graphique VRAM 64K ou 128K	256 × 192 points	Fonction palette de couleur, 16 couleurs/512 couleurs (2 couleurs/8 points)	—	Utilisé
3		64 × 48 points, multi-couleurs	Fonction palette de couleur, 16 couleurs/512 couleurs	—	Utilisé
4		256 × 192 points	Fonction palette de couleur, 16 couleurs/512 couleurs . (2 couleurs/8 points)	—	Utilisé (sprite amélioré)
5		256 × 212 points	Fonction palette de couleur, 16 couleurs/512 couleurs	2 pages (VRAM 64K) 4 pages (VRAM 128K)	Utilisé (sprite amélioré)
6		512 × 212 points	Fonction palette de couleur, 4 couleurs/512 couleurs	2 pages (VRAM 64K) 4 pages (VRAM 128K)	Utilisé (sprite amélioré)
7	Graphique VRAM 128K seulement	512 × 212 points	Fonction palette de couleur, 16 couleurs/512 couleurs	2 pages	Utilisé (sprite amélioré)
8		256 × 212 points	256 couleurs	2 pages	Utilisé (sprite amélioré)

Quel que soit le mode utilisé, les caractères et les figures sont affichés sur l'**avant-plan** de l'écran et, derrière celui-ci, se trouve **le fond**. Il est possible de changer la couleur du fond, mais pas d'y afficher des caractères ou des figures. Dans le haut et le bas de l'écran se trouve le **pourtour** dont, comme pour le fond, il est possible de changer la couleur, mais sur laquelle rien ne pourra être affiché.

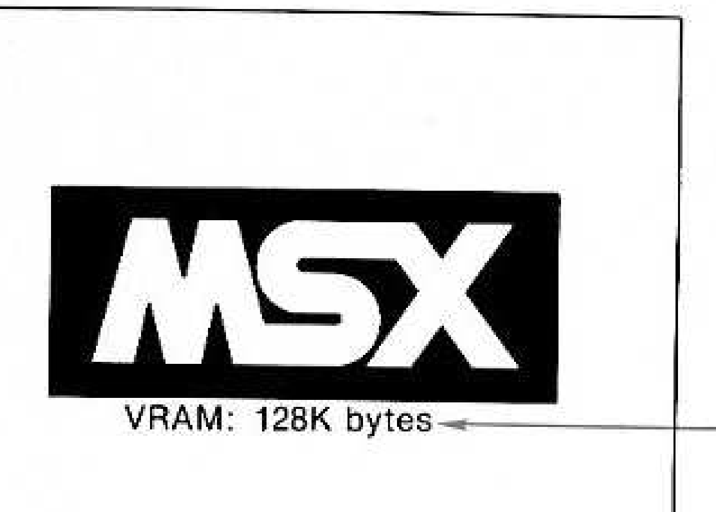
Devant l'avant-plan se trouvent **32 plans sprite** qui peuvent être utilisés pour afficher et animer des motifs sprite (ou des "lutins") en tout mode, à l'exception de SCREEN 0. Nous reviendrons, dans le chapitre suivant, sur les motifs sprite et leur utilisation.

Remarque sur la taille de la VRAM

Comme indiqué sur le tableau précédent, les modes SCREEN 7 et SCREEN 8 sont utilisés uniquement par des ordinateurs disposant d'une VRAM de 128K-octets. De plus, le nombre des pages, utilisables en modes SCREEN 5 et SCREEN 6, diffère d'après la capacité de la mémoire VRAM.

Le terme VRAM signifie Mémoire vive vidéo et il s'agit d'une mémoire qui conserve le contenu de ce qui est affiché sur l'écran.

La capacité de la VRAM de votre ordinateur est indiquée sur l'affichage du logo MSX, apparaissant lors de la mise en service du BASIC.



Affichage initial lors de la mise sous tension de l'ordinateur

MODE CARACTERE

Le mode caractère, destiné à l'affichage de textes sur l'écran, est spécifié par SCREEN 0 et SCREEN 1. En mode SCREEN 0, les caractères sont affichés par 6 points (largeur) × 8 points (hauteur), tandis qu'en mode SCREEN 1, l'affichage se fait en 8 points (largeur) × 8 points (hauteur). Comme la largeur de certains caractères graphiques des ordinateurs MSX se fait en 8 points, vous devrez recourir au mode SCREEN 1 pour afficher des caractères graphiques.

SPECIFICATION DU NOMBRE DE CARACTERES PAR LIGNE WIDTH

Le mode SCREEN 0 permet d'afficher jusqu'à 80 caractères par ligne, tandis que le nombre est limité à 32 par ligne dans le cas de SCREEN 1. **L'énoncé WIDTH sert à déterminer le nombre de caractères à afficher sur chaque ligne en mode texte.**

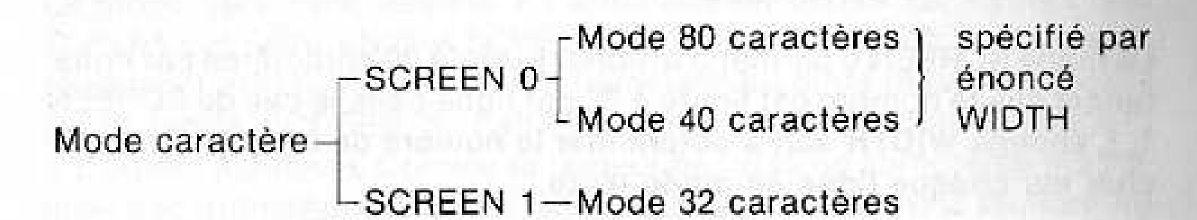
WIDTH nombre de caractères affichés

En mode SCREEN 0, la largeur des caractères affichés diffère selon que 1 à 40 caractères ou 41 à 80 caractères sont affichés par ligne.

A l'emploi de SCREEN 0: WIDTH 40 est exécuté.	A l'emploi de SCREEN 0: WIDTH 80 est exécuté.
ABCDEFG	ABCDEFG

Quand de 1 à 40 caractères sont spécifiés par ligne, on parle du **mode 40 caractères**. Par contre, on parle du **mode 80 caractères** quand de 41 à 80 caractères sont spécifiés.

Le nombre de caractères par ligne en mode SCREEN 1 va de 1 à 32 et la taille de tous est la même.



Lorsque WIDTH 80 est exécuté en mode SCREEN 0, 80 caractères, que WIDTH 40 est exécuté en mode SCREEN 0, 40 caractères et que WIDTH 32 est exécuté en mode SCREEN 1, le nombre des caractères remplit d'abord l'écran de gauche à droite; puis, à mesure que le nombre de caractères affichés sur chaque ligne décroît à chaque mode, la zone d'affichage des caractères est centrée au milieu de l'écran.

SCREEN 0: WIDTH 30	SCREEN 1: WIDTH 10
ABCDEFG	ABCDEFG

Le programme suivant illustre les sortes de changements que l'on peut obtenir dans le nombre de caractères, affichés par ligne.

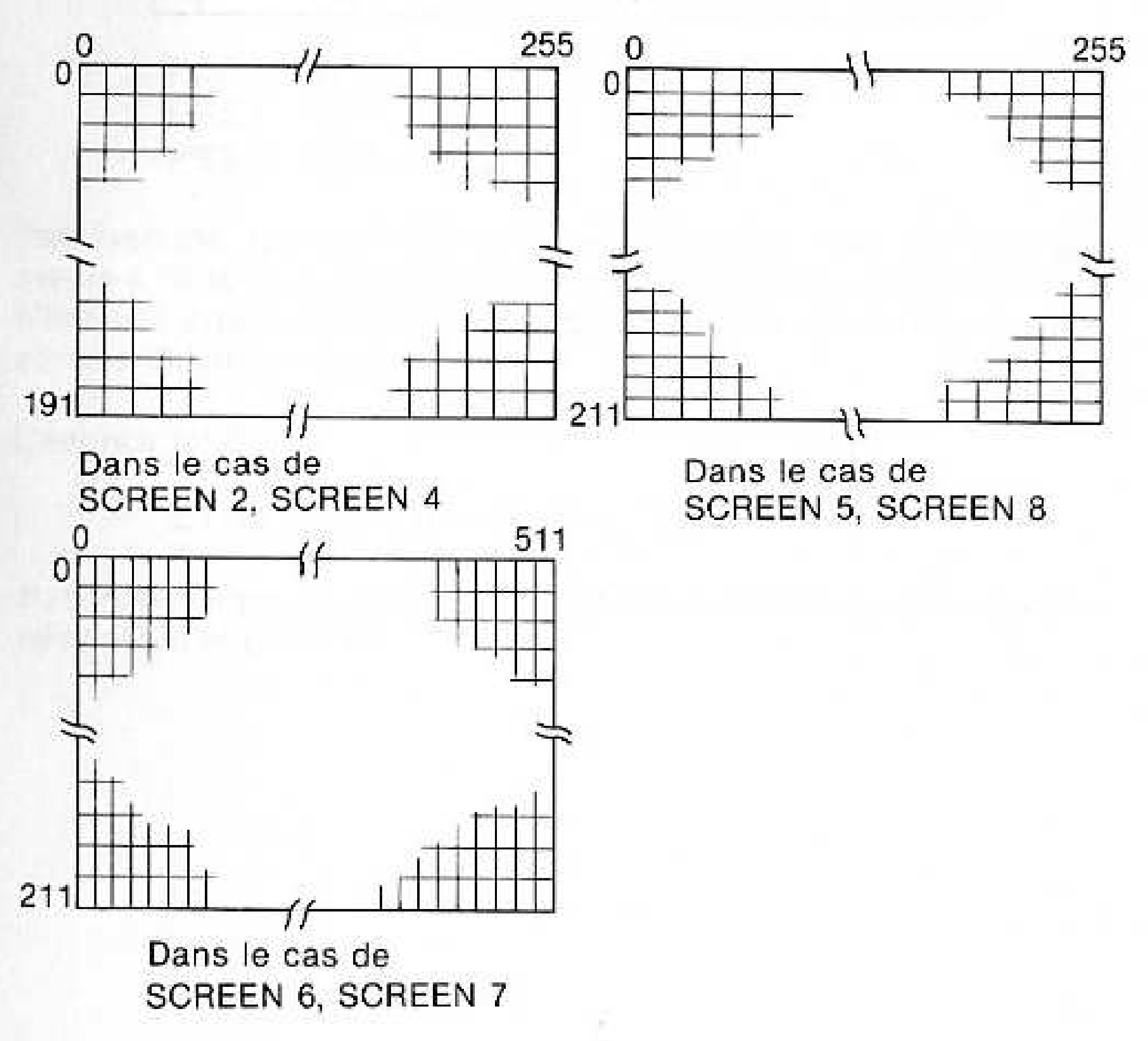
```
10 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 B$="abcdefghijklmnopqrstuvwxyz"
30 SCREEN 0:CLS
40 FOR W=80 TO 10 STEP -10
50 WIDTH W
60 GOSUB 150
70 NEXT W
80 SCREEN 1
90 FOR W=32 TO 12 STEP -10
100 WIDTH W
110 GOSUB 150
120 NEXT W
130 SCREEN 0:WIDTH 40
140 END
150 PRINT "WIDTH";W
160 PRINT:PRINT:PRINT
170 PRINT A$;B$
180 FOR T=0 TO 1000:NEXT T
190 RETURN
```

LE MODE GRAPHIQUE ET LES COORDONNEES

Les énoncés SCREEN 2 à SCREEN 8 spécifient les modes graphiques. Les énoncés suivants servent à tracer des figures dans tous les modes graphiques.

PSET, PRESET ... trace des points sur l'écran.
LINE ... trace des lignes et des rectangles.
CIRCLE ... trace des cercles, des ovales, des arcs et des formes en éventail.
PAINT ... colorie une figure à l'écran.
DRAW ... trace des figures, spécifiées par des sous-instructions graphiques.

L'écran se divise en coordonnées qui servent à spécifier des emplacements précis sur l'écran par les énoncés ci-dessus. Les coordonnées sont différentes selon les modes.



Dans le cas de SCREEN 2, SCREEN 4

Dans le cas de SCREEN 5, SCREEN 8

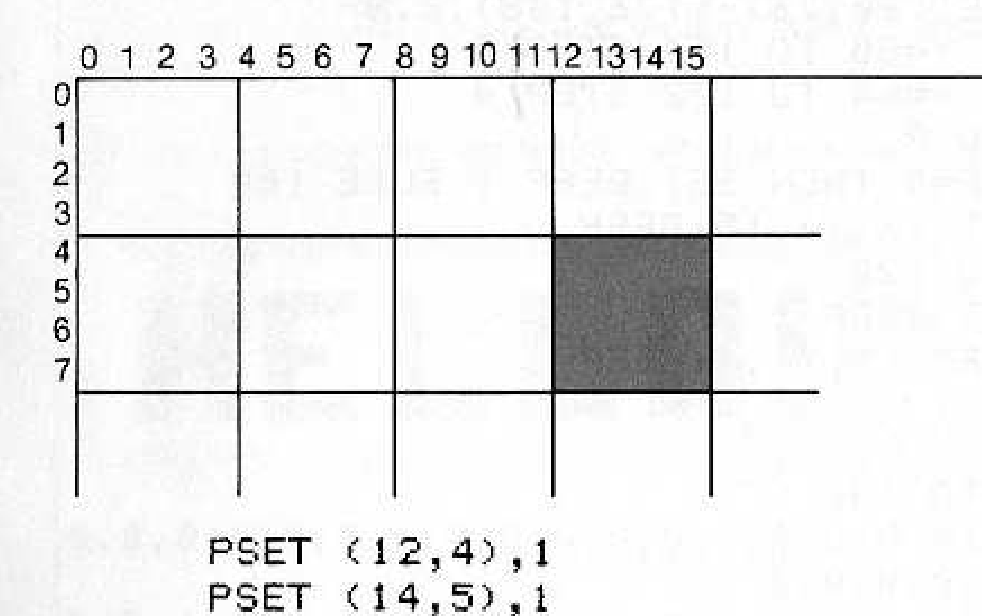
Dans le cas de SCREEN 6, SCREEN 7

Dans le programme suivant, les mêmes coordonnées sont utilisées en SCREEN 2, SCREEN 5 et SCREEN 6, mais le cercle est positionné en des endroits différents sur l'écran.

```
10 SCREEN 2
20 GOSUB 100
30 SCREEN 5
40 GOSUB 100
50 SCREEN 6
60 GOSUB 100
70 END
100 CIRCLE (125,100),90
110 FOR T=0 TO 1000:NEXT T
120 RETURN
```

LE MODE MULTI-COULEURS (SCREEN 3)

Les mêmes 256 × 192 coordonnées de points sont utilisées en mode SCREEN 3 que pour les modes SCREEN 2 et SCREEN 4, mais l'unité pour le tracé d'une figure est un bloc de 4 × 4 points dans ce cas.




```
PSET (12,4),1
PSET (14,5),1
PSET (15,7),1
```

Par exemple, les énoncés ci-dessus spécifient tous les emplacements à l'intérieur du même bloc de 4 × 4 points et, par conséquent, n'importe lequel de ces énoncés PSET coloriera tout le bloc en noir, comme illustré ci-dessus.

L'énoncé LINE

```
LINE (17,5)-(130,110)
```

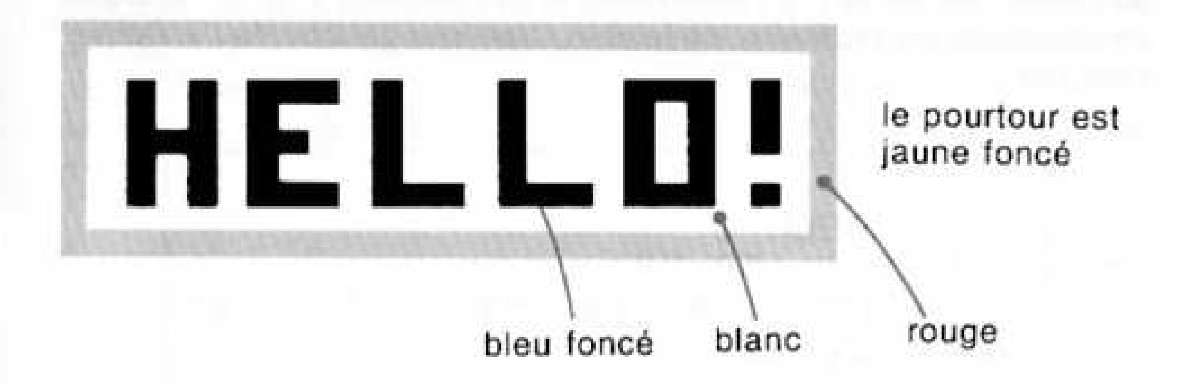
tracera une ligne grossière entre les blocs qui englobent les coordonnées (17,5) et (130,110).

Faisons tourner un programme qui utilise le mode multi-couleurs.

```
10 SCREEN 3
20 COLOR ,10,10:CLS
30 LINE (80,76)-(176,108),8,BF
40 FOR Y=80 TO 104 STEP 4
50 FOR X=84 TO 172 STEP 4
60 READ P
70 IF P=0 THEN SET BEEP 1 ELSE 100
80 PSET (X,Y),15:BEEP
90 GOTO 120
100 SET BEEP 2
110 PSET (X,Y),4:BEEP
120 NEXT X
130 NEXT Y
140 GOTO 140
150 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0
160 DATA 0,1,0,1,0,1,1,1,0,1,0,0,0,1,0,0
,0,1,1,1,0,1,0
170 DATA 0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,0
,0,1,0,1,0,1,0
180 DATA 0,1,1,1,0,1,1,0,0,1,0,0,0,1,0,0
,0,1,0,1,0,1,0
190 DATA 0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,0
,0,1,0,1,0,0,0
200 DATA 0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,1
,0,1,1,1,0,1,0
210 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0
```

L'énoncé SCREEN 3 à la ligne 10 spécifie le mode multi-couleur. Dans la boucle, l'énoncé READ de la ligne 60 affecte la donnée à la variable P. Si la valeur de P est 0, le point (bloc) tracé par l'énoncé PSET aura la couleur 15 (blanc). Si la valeur est différente de 0, la couleur sera 4 (bleu foncé). La valeur de P détermine aussi quel énoncé SET BEEP sera exécuté et, par conséquent, quel son sera spécifié. Ceci a pour résultat de produire un son différent chaque fois que l'énoncé PSET est exécuté.

Lorsque le programme est lancé, l'affichage suivant apparaît:



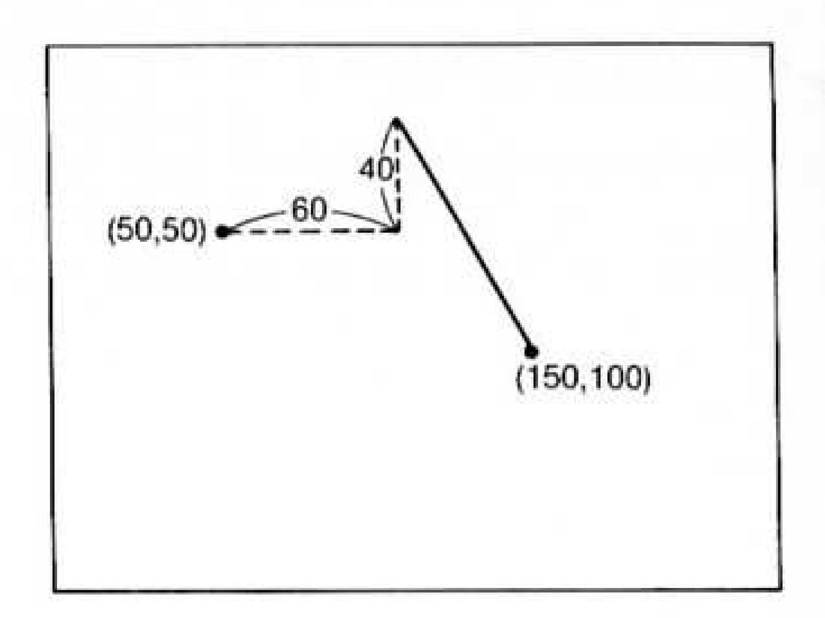
SPECIFICATION STEP

La spécification STEP (X,Y) peut être utilisée avec les énoncés PSET, PRESET, LINE, CIRCLE, PAINT et PUT SPRITE (expliqué dans le chapitre suivant) afin de spécifier les coordonnées (X,Y).

Quand ces énoncés graphiques sont exécutés, le dernier point spécifié est mémorisé. Si STEP (X,Y) est exécuté ensuite, l'emplacement de (X,Y) est déterminé sur un nouveau système de coordonnées et le point spécifié en dernier lieu devient l'origine (0,0). Si la spécification STEP est omise, les emplacements sont spécifiés selon le système de coordonnées ordinaire, où le coin supérieur gauche de l'écran est l'origine.

```
10 SCREEN 2
20 PSET (50,50)
30 LINE STEP (60,-40)-(150,100)
40 GOTO 40
```

Dans ce programme, les coordonnées (50,50), spécifiées par l'énoncé PSET de la ligne 20, sont mémorisées. Ensuite, à la ligne 30, STEP (60, − 40) est utilisé pour spécifier le point de départ pour l'énoncé LINE. Les coordonnées (50,50) deviennent alors la nouvelle origine et l'emplacement 60 sur l'axe X et − 40 sur l'axe Y à compter de la nouvelle origine, devient le point de départ pour la ligne.



Les formats suivants sont utilisés quand STEP est inclus dans les énoncés graphiques.

PSET STEP(X,Y), couleur
PRESET STEP(X,Y), couleur
LINE STEP(X,Y)—STEP(X,Y), couleur, B / BF
CIRCLE STEP(X,Y), rayon, couleur, angle départ, angle fin, rapport de format

Affichage des caractères dans les modes graphiques

Des caractères peuvent aussi être affichés sur l'écran en mode graphique. A cet effet, l'affichage graphique est utilisé comme périphérique de fichiers. Le fichier est ouvert et les caractères à afficher sont sortis vers le fichier. Voir en page 250 pour un complément d'informations.

SPECIFICATION DES COULEURS

- La fonction palette
- Spécification de la palette—COLOR
- Le mode SCREEN 8 et les couleurs
- Epanchement de couleur (SCREEN 2 et SCREEN 4)
- Retour des spécifications de couleur aux réglages initiaux—COLOR

LE CODE COULEUR ET LA FONCTION PALETTE

Dans le mode SCREEN 2, 16 couleurs peuvent être utilisées pour tracer des graphiques, repérée chacune par un code. Voici la liste des codes de couleur:

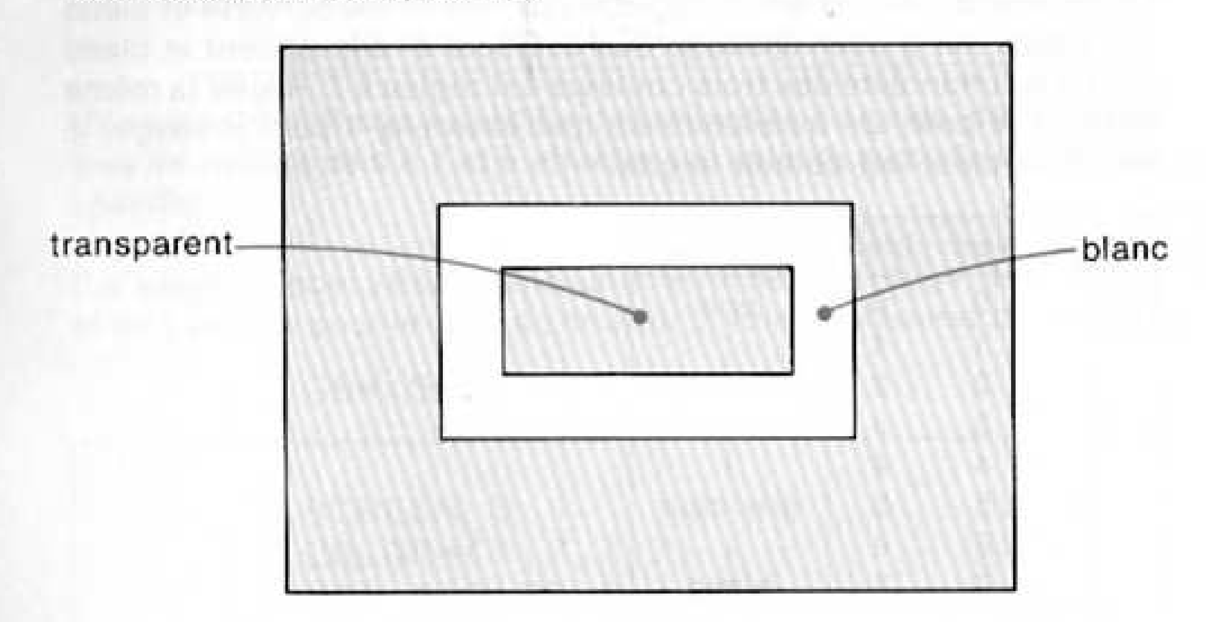
Tableau des codes de couleur

Code	Couleur	Code	Couleur	Code	Couleur	Code	Couleur
0	transparent	4	bleu foncé	8	rouge médium	12	vert foncé
1	noir	5	bleu clair	9	rouge clair	13	magenta
2	vert médium	6	rouge foncé	10	jaune foncé	14	gris
3	vert clair	7	bleu ciel	11	jaune clair	15	blanc

La couleur transparente
Si l'on spécifie le code de couleur 0, la couleur est le transparent, ce qui signifie que si la couleur transparente est utilisée pour tracer une figure sur l'avant-plan, la couleur du fond apparaîtra à travers la figure. Ceci peut se vérifier par le programme suivant.

```
10 SCREEN 2
20 FOR B=2 TO 14
30 COLOR ,B,B:CLS
40 LINE (50,50)-(180,140),15,BF
50 LINE (70,70)-(160,120),0,BF
60 FOR T=0 TO 1000:NEXT T
70 NEXT B
80 COLOR ,4,4:CLS
90 END
```

La boucle FOR—NEXT change successivement la couleur du fond et celle du pourtour de 2 (vert médium) en 14 (gris). Un carré blanc est tracé (ligne 40) et, à l'intérieur de celui-ci est tracé un carré transparent plus petit (ligne 50).
Ensuite, chaque fois que change la couleur du fond, le carré transparent prend la même couleur, puisque la couleur du fond passe à travers la couleur transparente.



La palette des couleurs
Comme on l'a vu à la page 94, les modes, utilisant les 10 couleurs du code 0 au code 15, sont: SCREEN **0**, SCREEN **1**, SCREEN **2**, SCREEN **3**, SCREEN **4**, SCREEN **5**, et SCREEN **7**.
Les 16 couleurs, présentées dans le tableau ci-dessus, sont les 16 couleurs utilisables à la mise en service du BASIC. Mais il ne s'agit nullement là d'une liste exhaustive des couleurs, disponibles en MSX2 car, en réalité, les codes de 0 à 15 autorisent la création de 512 couleurs différentes.
Comme il serait impossible de nommer toutes ces 512 nuances colorées, les couleurs sont spécifiées en déterminant le degré de rouge, de vert et de bleu qui sera mélangé pour produire une couleur donnée. Ainsi, on parlera de rouge 3, de vert 2 et de bleu 7, un peu comme un peintre mélange plusieurs mesures de rouge, de vert et de bleu respectivement sur sa palette. On appelle cela la **fonction palette**.

Remarque
Les couleurs peuvent être spécifiées pour le code de couleur 0, mais il s'agit là d'un cas particulier. Nous expliquerons ici l'emploi des codes de couleur de 1 à 15 pour créer des couleurs différentes.

UTILISATION DE LA FONCTION PALETTE

Les trois couleurs rouge, vert et bleu ont des niveaux de clarté, allant de 0 à 7. Ces niveaux sont appelés la **luminosité**. Comme, pour chaque couleur, il existe 8 niveaux différents, ceux-ci peuvent être combinés pour créer 512 couleurs, soit $8 \times 8 \times 8 = 512$.
La couleur devient le noir lorsque la luminosité rouge, verte et bleue est réglée sur 0 pour chacune des couleurs et elle devient le blanc quand la luminosité des trois couleurs est réglée à 7. Régler la même luminosité pour les trois couleurs (par exemple, 4 pour le rouge, le vert et le bleu) fait obtenir le gris.

rouge	vert	bleu	couleur
0	0	0	noir
1	1	1	↑
2	2	2	gris foncé
3	3	3	↕
4	4	4	
5	5	5	gris clair
6	6	6	↓
7	7	7	blanc

Lorsque la luminosité d'une des couleurs est rendue plus grande que celle des deux autres, (ou plus proche est de 0 le réglage de luminosité des deux autres couleurs) la couleur la plus claire prédominera. Par exemple, 7, 0, 0 produira un rouge pur, comme le montrent les exemples de réglage de luminosité ci-après.

rouge	vert	bleu	couleur
4	3	3	gris, avec une légère teinte rouge
5	2	2	une couleur proche du rouge
5	0	0	rouge (légèrement foncé)
7	0	0	rouge (rouge pur, la couleur la plus vive)
2	0	0	rouge, presque noir

SPECIFICATION DE LA PALETTE COLOR

L'énoncé COLOR s'emploie pour spécifier le code de couleur et les réglages de luminosité nécessaires pour obtenir la couleur souhaitée.

COLOR = (code couleur, luminosité du rouge, luminosité du vert, luminosité du bleu)

L'énoncé COLOR sert à spécifier la luminosité du rouge, du vert et du bleu en valeurs de 0 à 7 et à affecter ces valeurs au code de couleur spécifié.

Par exemple, pour affecter une luminosité de 4 au rouge, de 3 au vert et de 1 au bleu pour le code de couleur 5, vous exécuterez ce qui suit:

```
COLOR=(5,4,3,1)
```

```
10 SCREEN 5
20 COLOR=(1,7,7,7)
30 FOR C=2 TO 9
40 COLOR=(C,0,0,C-2)
50 NEXT C
60 COLOR ,1,1:CLS
70 FOR CC=2 TO 9
80 R=100-CC*10
90 CIRCLE (125,100),R,CC
100 PAINT (125,95),CC
110 NEXT CC
120 GOTO 120
```

Les couleurs pour les codes couleur de 1 à 9 sont spécifiés dans les lignes 20 à 50 dans ce programme.

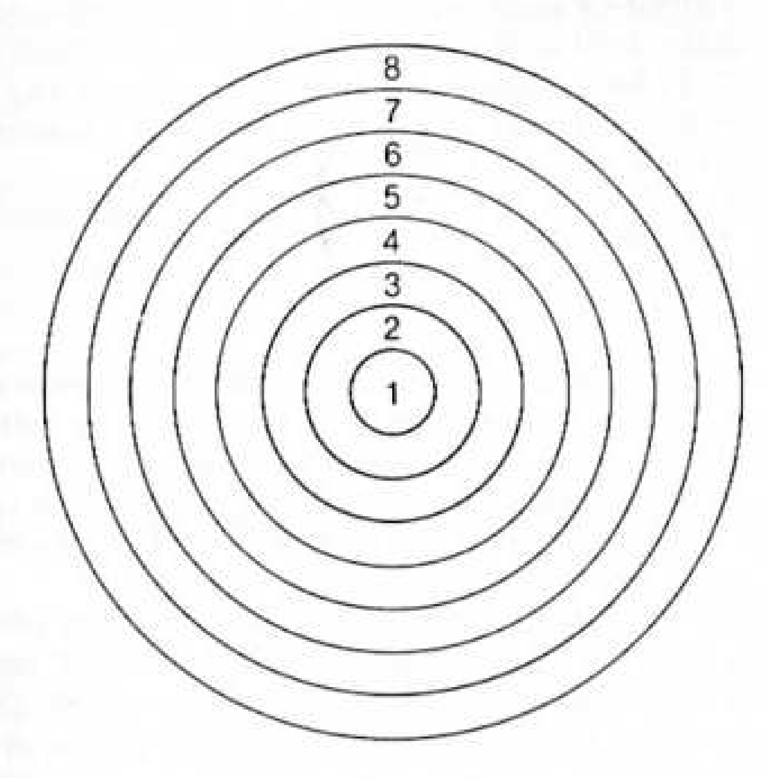
code	rouge	vert	bleu
1	7	7	7
2	0	0	0
3	0	0	1
4	0	0	2
5	0	0	3
6	0	0	4
7	0	0	5
8	0	0	6
9	0	0	7

Le code 1 est le blanc, le code 2 le noir et les codes de 3 à 9 changent la couleur par paliers du bleu presque noir au bleu pur.
Dans l'énoncé COLOR de la ligne 60, le code 1 est spécifié comme couleur du fond et du pourtour, mais le code couleur est à présent le blanc au lieu du noir. A partir de la ligne 70, des cercles de différentes nuances de bleu sont tracés de façon concentrique vers le centre, le cercle final étant tracé en bleu pur.
La fonction palette peut être utilisée de cette façon pour réaliser un dessin, présentant différentes nuances de la même couleur.
Le programme précédent utilise le mode SCREEN 5, le plus approprié pour réaliser rapidement des affichages graphiques. La vitesse de tracé des dessins par énoncés graphiques en modes SCREEN 2 et SCREEN 4 est plus lente et il y a, en outre, un risque d'épanchement, un phénomène qui sera expliqué dans la section suivante.

Rédigeons un autre programme, faisant appel à la fonction palette.

```
10 SCREEN 5
20 FOR L=8 TO 1 STEP -1
30 CIRCLE (120,100),L*10+5,L
40 PAINT (120,100),L,L
50 NEXT L
60 K=(K+1) MOD 8
70 FOR L=1 TO 8
80 COLOR=(L,K,K,0)
90 K=(K+1) MOD 8
100 NEXT L
110 GOTO 60
```

L'illustration suivante indique les codes couleur pour les cercles concentriques, tracés par la boucle FOR—NEXT des lignes 20 a 50.



Tout d'abord, les couleurs correspondant aux codes couleur de 1 à 8 sont noir, vert médium, vert clair, bleu foncé, bleu clair, rouge foncé, bleu ciel et rouge médium. Les cercles sont tracés dans ces couleurs. Ensuite, la fonction palette est utilisée pour changer les couleurs, correspondant aux numéros de code. Ceci est accompli aux lignes de 60 à 110.

Dans l'énoncé COLOR de la ligne 80, la variable K est utilisée pour spécifier la luminosité du rouge et celle du vert. La valeur de K est définie aux lignes 60 à 90. Le MOD utilisé dans ces lignes est un des quatre signes arithmétiques, tels que +, −, * et /.

a + b ajoutera a et b, mais **a MOD b donne le reste de la division de a par b.** Le tableau suivant indique la relation entre K et (K + 1), (K + 1) MOD 8.

K	K + 1	(K + 1) MOD 8
0	1	1 (1 ÷ 8 = 0 ... 1)
1	2	2 (2 ÷ 8 = 0 ... 2)
2	3	3 (3 ÷ 8 = 0 ... 3)
3	4	4 (4 ÷ 8 = 0 ... 4)
4	5	5 (5 ÷ 8 = 0 ... 5)
5	6	6 (6 ÷ 8 = 0 ... 6)
6	7	7 (7 ÷ 8 = 0 ... 7)
7	8	0 (8 ÷ 8 = 1 ... 0)

Si aucune valeur n'est affectée à une variable, sa valeur sera 0. En conséquence, la valeur de K avant que la ligne 60 ne soit exécutée est 0, et, quand la ligne est exécutée, elle devient 1. Ensuite, chaque fois que K = (K + 1) MOD 8 est exécuté à la ligne 90 et à la ligne 60, la valeur de K est augmentée de 1. Après que K est devenu 7, elle devient 0, puis elle augmente à nouveau jusqu'à 7.
Ainsi, par suite des changements de la valeur de K et en raison de l'énoncé COLOR à l'intérieur de la boucle FOR—NEXT, les couleurs des codes de 1 à 8 sont changées par les modifications des niveaux de luminosité du rouge et du vert de 0 à 7. Dans tout le programme, le niveau de luminosité du bleu reste à 0.

Si la ligne 80 est modifiée comme suit:

```
80 COLOR=(L,0,K,K)
```

le niveau de luminosité du rouge reste à 0, tandis que les niveaux du vert et du bleu changent. Le vert peut également être fixé à 0 et la luminosité de deux autres couleurs changée. Essayez les diverses combinaisons et observez les résultats obtenus sur l'écran.

LE MODE SCREEN 6 ET LA FONCTION PALETTE

La fonction palette peut servir aussi en mode SCREEN 6, mais seuls les codes couleur de 0 à 3 sont utilisables, soit seulement 4 des 512 couleurs. Les couleurs de SCREEN 6 sont définies comme sur le tableau suivant à la mise en service du BASIC.

code	couleur
0	transparent
1	noir
2	vert
3	vert clair

LE MODE SCREEN 8 ET LES COULEURS

La fonction palette n'est pas utilisée en mode SCREEN 8, mais 256 couleurs sont disponibles en faisant appel aux codes couleur.
En mode SCREEN 8, le rouge et le vert ont respectivement 8 niveaux de luminosité, allant de 0 à 7, tandis qu'il existe 4 niveaux de luminosité du bleu, de 0 à 3. Comme $8 \times 8 \times 4 = 256$, les codes couleur de 0 à 255 sont utilisés. Un code couleur est déterminé par la formule suivante:

$$\textbf{code couleur} = 32 \times \textbf{(luminosité vert)} + 4 \times \textbf{(luminosité rouge)} + \textbf{(luminosité bleu)}$$

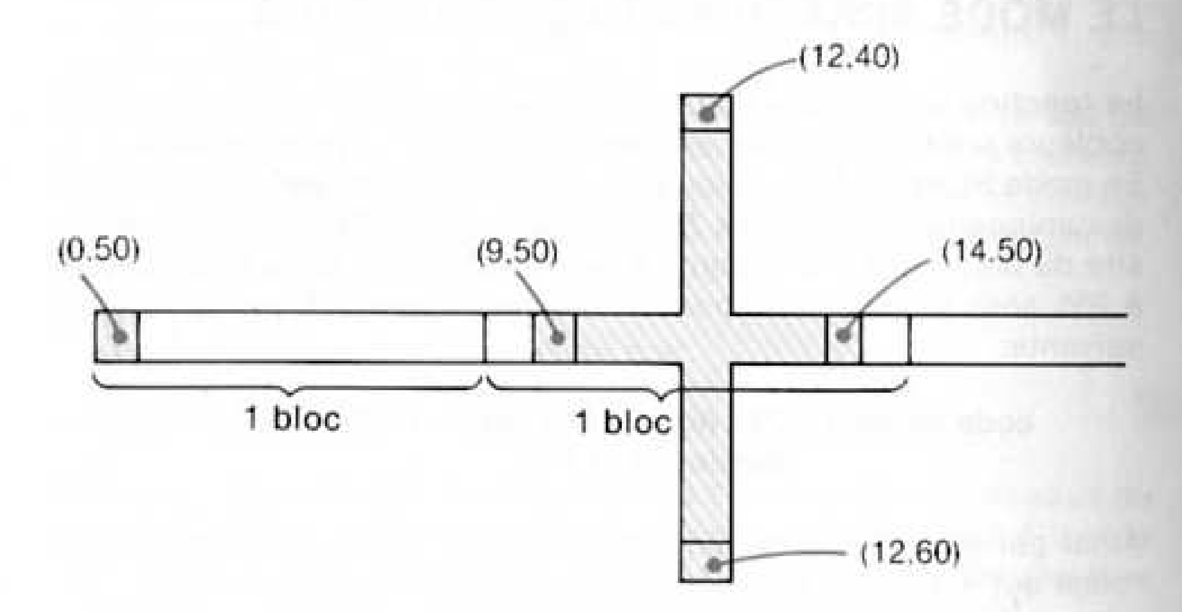
Ainsi par exemple, si la luminosité du vert est 1, la luminosité du rouge est 4 et celle du bleu est 3, le code de couleur sera 55:

$$32 \times 1 + 4 \times 5 + 3 = 55$$

EPANCHEMENT DE COULEUR EN MODES SCREEN 2 et SCREEN 4

En modes SCREEN 2 et SCREEN 4, seules 2 couleurs (y compris la couleur du fond) peuvent être spécifiées pour un bloc de 8 points horizontaux. Si plus de 2 couleurs sont spécifiées, celle qui l'a été en dernier lieu devient la couleur valide.

```
10 SCREEN 2
20 LINE (9,50)-(14,50),15
30 LINE (12,40)-(12,60),1
40 GOTO 40
```



Dans ce programme, une ligne horizontale est tracée de X9 à X14 dans le bloc horizontal de 8 points, allant de X8 à X15. Ensuite, une ligne verticale est tracée en X12, de Y40 à Y60, et celle-ci coupe la ligne horizontale. Ceci ajoute une troisième couleur au bloc horizontal de X8 à X15; par conséquent, même si l'on spécifie le blanc, la ligne horizontale sera affichée en noir, étant donné que le noir, la dernière couleur spécifiée, devient la couleur valide pour ce bloc.
Lorsque, de cette façon, la couleur spécifiée devient une couleur différente, on parle d'un **épanchement de couleur**. Pour éviter cet inconvénient, une certaine prudence est donc de rigueur lors de l'affectation des couleurs en modes SCREEN 2 et SCREEN 4.
Si la ligne 20 est modifiée comme suit:

```
LINE (8,50)-(15,50)
```

la ligne horizontale remplira tout le bloc de huit points, de X8 à X15, et la couleur spécifiée (le blanc) restera valide.
Toutefois, si une autre couleur est spécifiée par la suite pour le même bloc, par un énoncé tel que

```
PSET (8,50),8
```

la couleur de tout le bloc sera changée en la couleur spécifiée en dernier lieu (dans le cas de l'énoncé PSET, le code couleur 8).

Se souvenir toujours qu'en modes SCREEN 2 et SCREEN 4, un maximum de 2 couleurs est utilisable en tout bloc de 8 points.

Dans les modes SCREEN de 5 à 8, des couleurs peuvent être spécifiées à volonté en unité de 1 point chacune.

RETOUR DES SPECIFICATIONS DE COULEUR AUX REGLAGES INITIAUX COLOR

Les codes couleur et les spécifications de couleur ont été changés par l'énoncé COLOR. Pour ramener les spécifications dans un programme à leur état initial lors de la mise en service du BASIC

COLOR = NEW

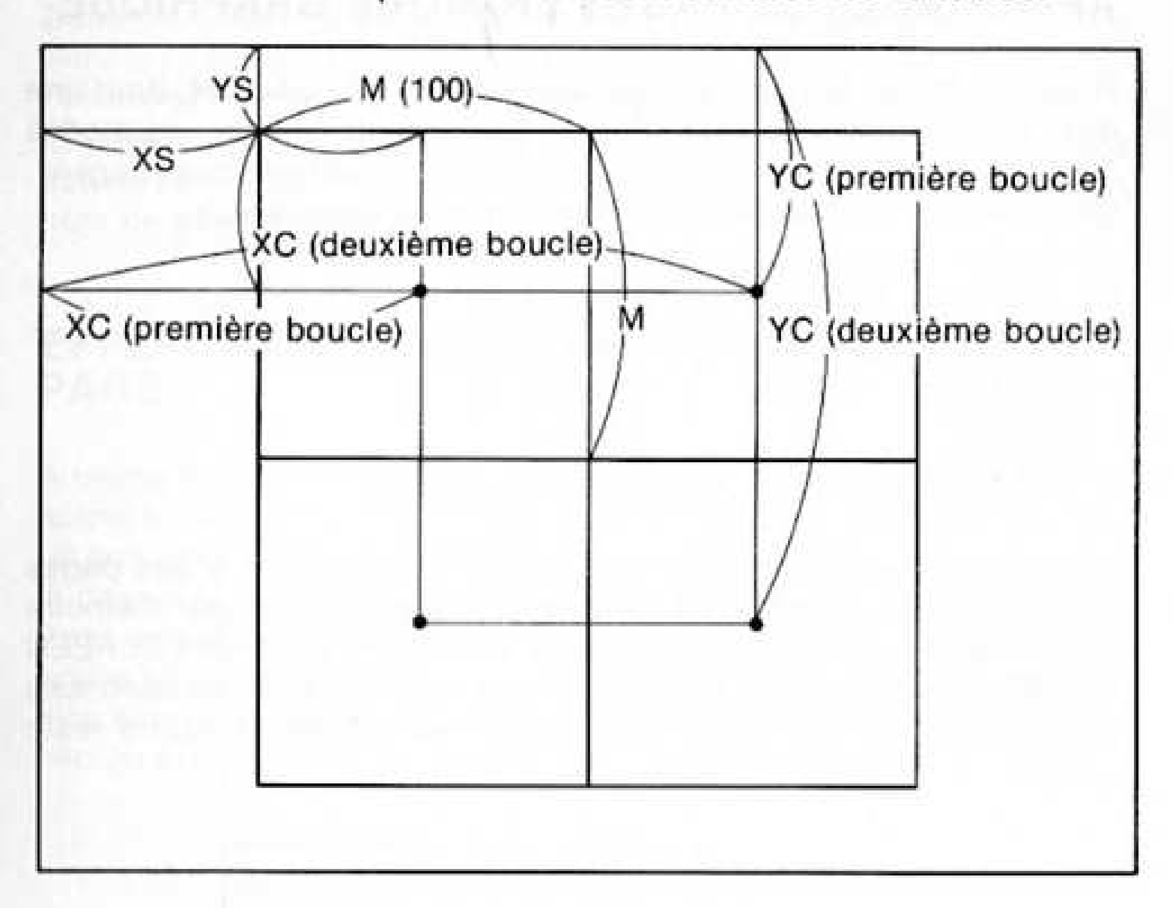
est exécuté.

Exemple de programme

Le programme suivant utilise la fonction palette en mode SCREEN 5.

```
10 S=2:CN=10:L=50:M=L*2
20 XS=(255 MOD M)/2:YS=(211 MOD M)/2
30 COLOR 15,0,0:SCREEN 5
40 '-
50 FOR T=0 TO TIME-INT(TIME/100)*100:J=R
ND(1):NEXT
60 ' *** draw box ***
70 FOR XC=L+XS TO 255-L STEP M
80  FOR YC=L+YS TO 211-L STEP M
90  C=0
100   FOR P=L TO 0 STEP -S
110    LINE(XC-P,YC-P)-STEP(P*2,P*2),C+1
,BF
120    C=(C+1)MOD CN
130   NEXT P
140  NEXT YC
150 NEXT XC
160 ' *** define color ***
170 R=RND(1)*5+2:G=RND(1)*5+2:B=RND(1)*5
+2
180 FOR P=1 TO CN
190  J=P/CN:R(P)=R*J:G(P)=G*J:B(P)=B*J
200 NEXT P
210 ' *** change color ***
220 FOR K=0 TO 20
230  FOR P=1 TO CN
240   COLOR=(J+1,R(P),G(P),B(P))
250   J=(J+1) MOD CN
260  NEXT P
270  J=(J+1) MOD CN
280 NEXT K
290 GOTO 170
```

Dans ce programme, on remarquera certaines instructions qui n'ont pas encore été expliquées; pour l'instant, veuillez les entrer telles quelles et lancer le programme.
Voici une brève description du programme.
Tout d'abord, les lignes 70 à 150 tracent des carrés. Les variables utilisées dans ces lignes sont définies dans les lignes 10 et 20. L'illustration suivante indique comment sont utilisées les variables.



Les variables en tableau R(P), G(P), et B(P) sont utilisées aux lignes 170 à 200. Les valeurs de 2 à 7 sont affectées à R(1)—R(10), G(1)—G(10) et B(1)—B(10). La fonction RND de la ligne 170 détermine quelle valeur sera affectée. La fonction RND procure un nombre positif supérieur à 0 et inférieur à 1. Ces fonctions seront expliquées au Chapitre 7.
Aux lignes 220 à 280, les valeurs de R(P), G(P), et B(P) sont utilisées pour changer les couleurs des codes couleur 1 à 10, en faisant appel à la fonction palette.

L'apostrophe (') des lignes 40, 60, 160 et 210 est utilisée à la place de REM. **L'énoncé REM ou (') s'emploie pour écrire, dans un programme, une remarque qui ne sera pas exécutée comme une partie de celui-ci.**

DEFINITION DES PAGES

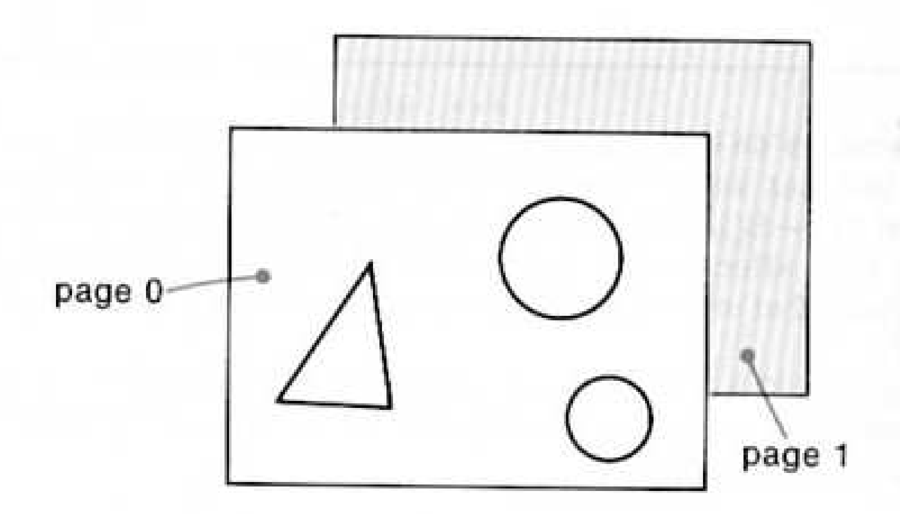
- A propos des pages
- Page d'affichage et page active
- Définition des pages—SET PAGE

AFFICHAGES ET PAGES EN MODE GRAPHIQUE

Retournons voir le tableau de mode SCREEN à la page 94, dont une des colonnes porte le titre "Page". En mode graphique, les modes SCREEN de 5 à 8 utilisent, en effet, ce que l'on appelle des "pages". Cette portion du tableau peut être transcrite comme suit:

Mode	VRAM 64K	VRAM 128K
SCREEN 5	2 pages	4 pages
SCREEN 6	2 pages	4 pages
SCREEN 7	—	2 pages
SCREEN 8	—	2 pages

Comme leur nom l'indique, ces "pages" ressemblent à des pages d'un carnet de notes. Des ordinateurs, disposant d'une mémoire VRAM de 64K-octets, ont deux pages utilisables en modes SCREEN 5 et SCREEN 6. Quand on trace un dessin sur l'écran par des énoncés graphiques, une seule des deux pages est utilisée et l'autre reste vierge.



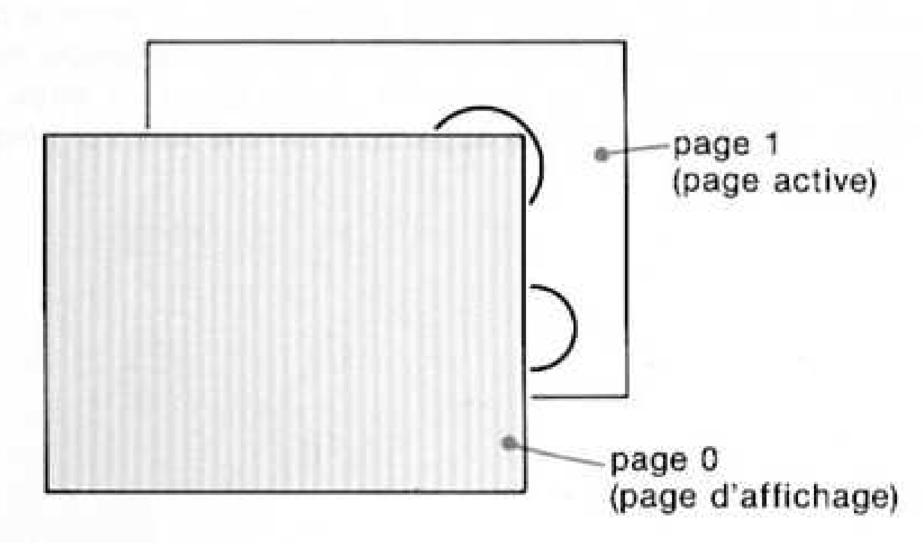
Pour une VRAM de 64K, en SCREEN 5, SCREEN 6

Comme le montre cette illustration, lorsque les deux pages sont utilisées, elles reçoivent un **numéro de page**. L'une est appelée page 0 et l'autre, page 1. Avec une mémoire VRAM de 128 K-octets, quatre pages sont disponibles en modes SCREEN 5 et SCREEN 6 et elles sont alors appelées respectivement page 0, page 1, page 2 et page 3.

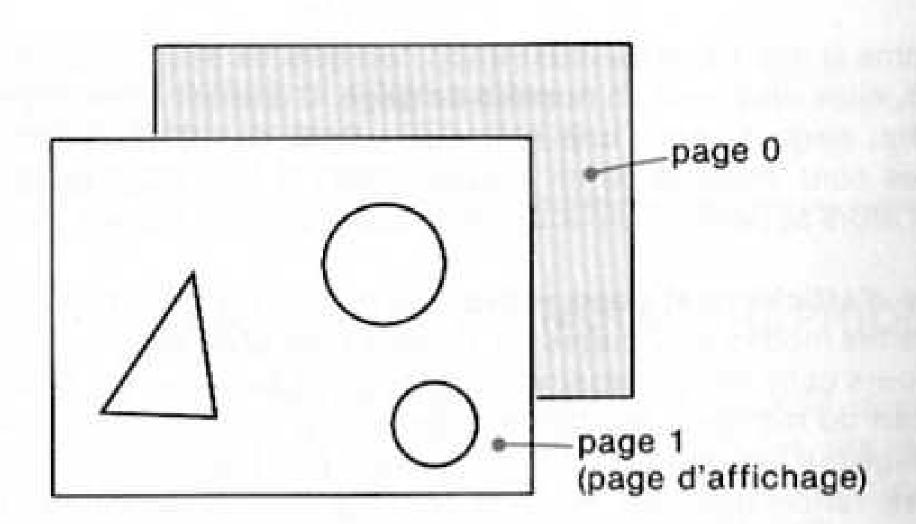
Page d'affichage et page active
Dans les modes où 2 pages ou 4 pages sont utilisables, la page 0 est toujours celle où les dessins sont tracés et celle qui est affichée sur l'écran du moniteur au moment de la mise en service du BASIC.
La page sur laquelle les dessins peuvent être tracés est appelée **page active**, tandis que celle que l'on peut voir sur l'écran de visu porte le nom de **page d'affichage**.

EFFETS DISPONIBLES PAR DEFINITION DE LA PAGE

A moins de spécifications contraires, la page d'affichage et la page active sont toutes deux la page 0. Par conséquent, quand un énoncé, tel que LINE, CIRCLE ou DRAW, est exécuté, le dessin sera tracé sur la page 0 et il sera affiché tel quel sur l'écran. Cependant, si la page 0 est déterminée comme page d'affichage et la page 1 comme page active et que des énoncés graphiques sont alors exécutés, les dessins seront tracés sur la page 1, mais ceux-ci n'apparaîtront pas sur l'écran étant donné que ce que l'on voit sur l'écran est la page 0.



Si l'on fait de la page 1 la page d'affichage, les dessins qui y seront tracés seront affichés sur l'écran.

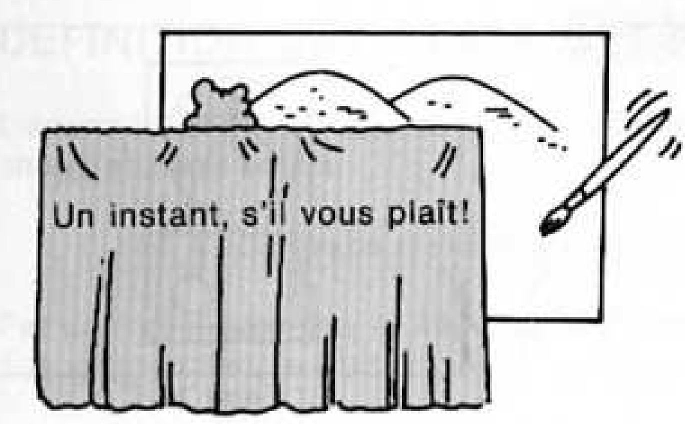


A ce moment, si la page 0 est choisie comme page active, les figures ultérieures seront tracées sur la page 0. Et si l'on fait de la page 1 la page active, les dessins tracés ultérieurement seront ajoutés à la page 1.

Utilisation des pages

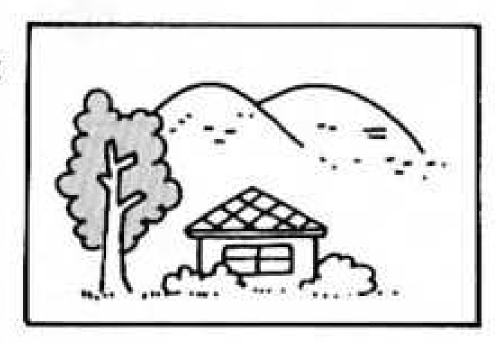
Il existe de nombreuses façons d'utiliser la fonction de définition des pages. Voici, à titre d'exemple, deux effets intéressants que vous pourrez obtenir.

- **Présentation sur l'écran du dessin après sa réalisation complète**
  Un certain temps est nécessaire pour terminer un dessin complexe, surtout si vous faites appel à de nombreux énoncés PAINT. Si vous ne désirez pas montrer le dessin pendant qu'il est tracé sur l'écran, vous pourrez déterminer des pages différentes comme page d'affichage et page active et tracer alors le dessin en toute tranquillité sur la page active. Lorsque le dessin sera terminé, il vous suffira de changer la page active en page d'affichage et le dessin achevé apparaîtra en une fois sur l'écran.

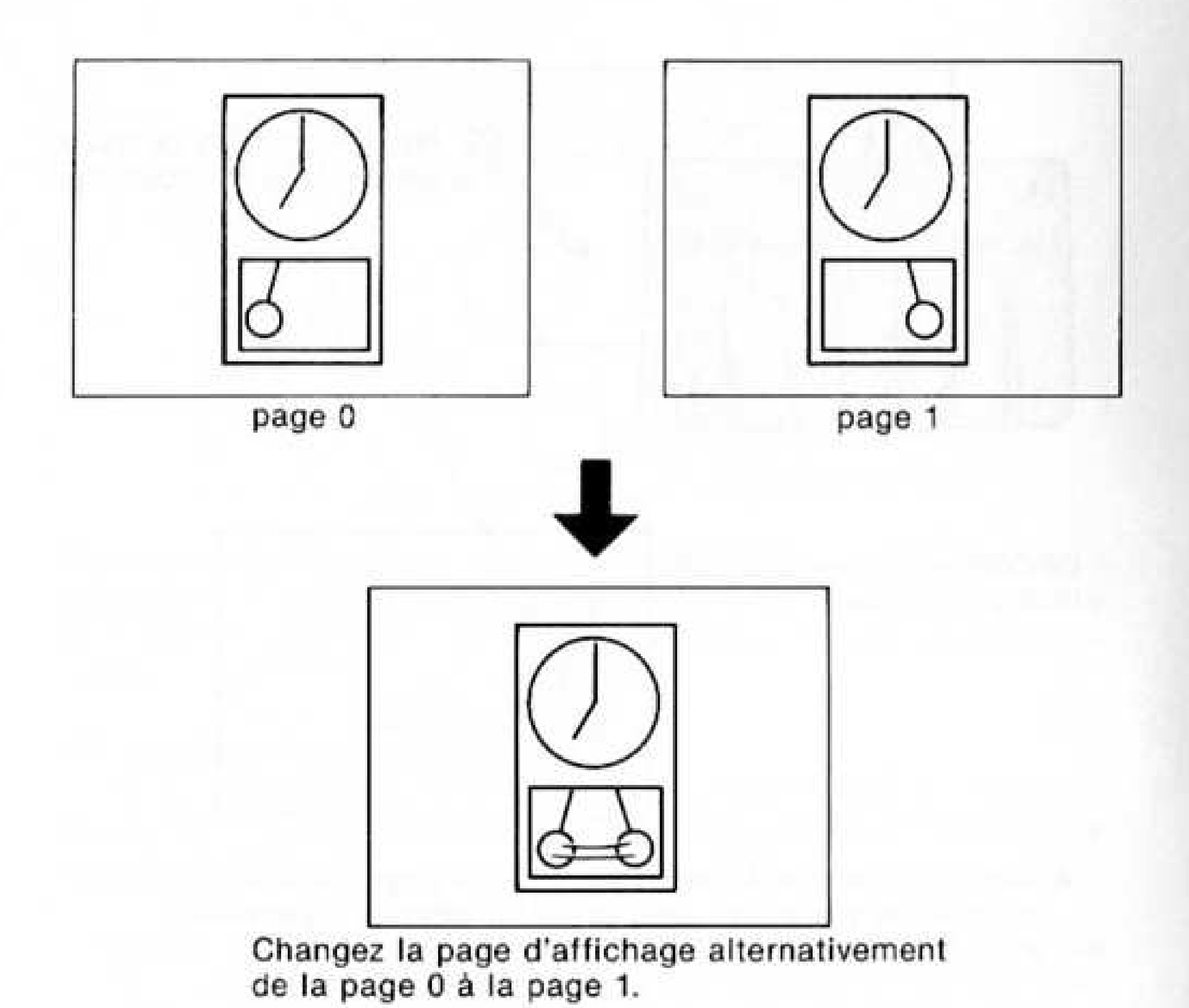


Ne montrez pas ce qui se passe dans les coulisses!

Le dessin achevé apparaît tout d'un coup!



- **Tracé d'un dessin différent sur chaque page, suivi d'une permutation entre les pages pour créer un effet de mouvement**
  Vous pourrez obtenir l'effet suivant, par exemple:



Changez la page d'affichage alternativement de la page 0 à la page 1.

DEFINITION DES PAGES SET PAGE

L'énoncé SET PAGE détermine les numéros des pages: page d'affichage et page active.

SET PAGE [page d'affichage], [page active]

Par exemple, pour que la page 0 soit la page d'affichage et la page 1 la page active, exécutez:

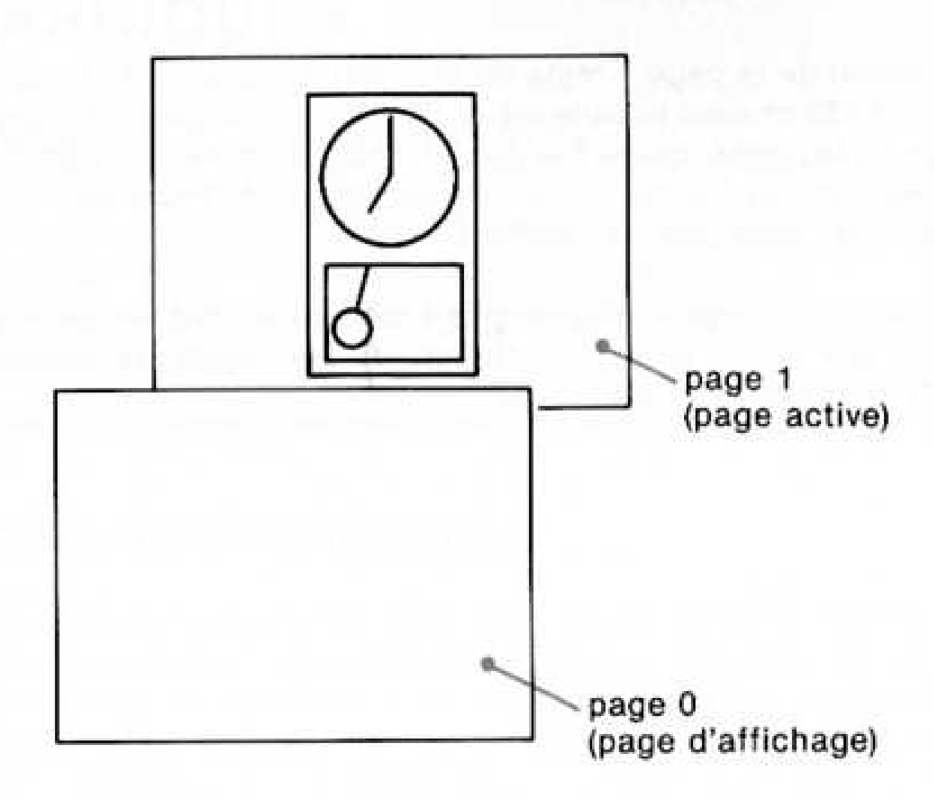
```
SET PAGE 0,1
```

Quand cet énoncé SET PAGE est exécuté, la page 0 est celle qui est visible sur l'écran, tandis que la page 1 est celle où sont tracés les dessins.

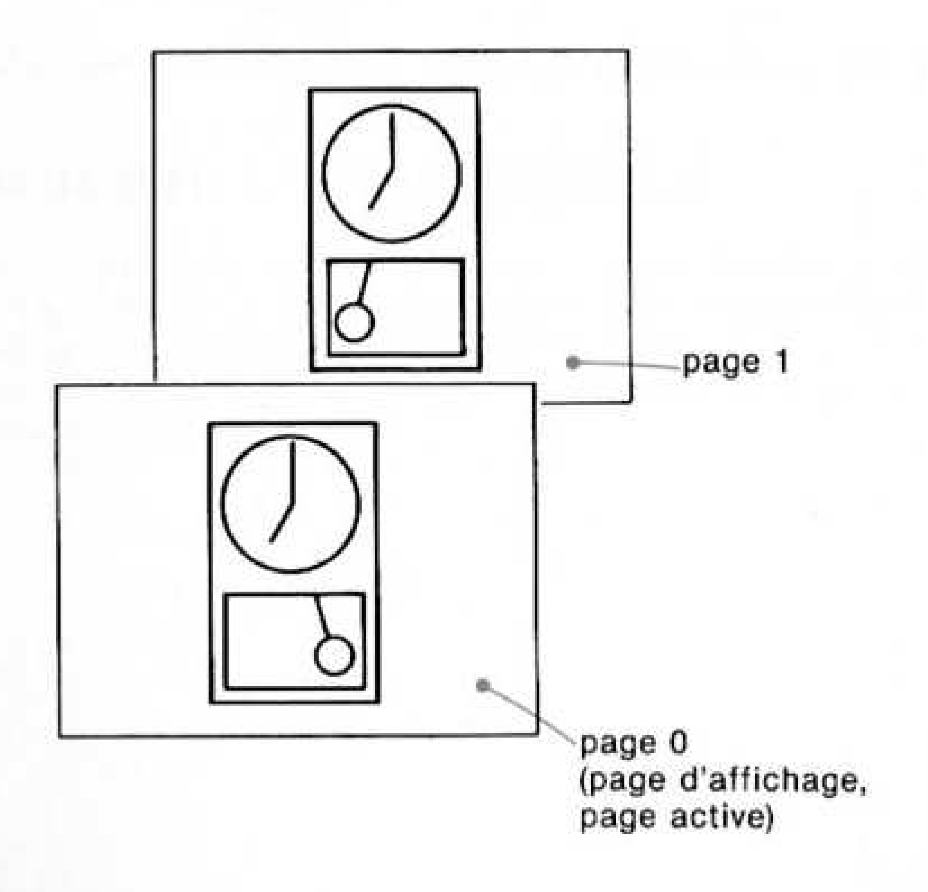
Changement des pages
Le programme suivant fait permuter alternativement les pages.

```
10 SCREEN 5
20 SET PAGE 0,1:CLS
30 X1=110:X2=100
40 GOSUB 130
50 SET PAGE 0,0:CLS
60 X1=140:X2=150
70 GOSUB 130
80 SET PAGE 1
90 FOR T=0 TO 300:NEXT T
100 SET PAGE 0
110 FOR T=0 TO 300:NEXT T
120 GOTO 80
130 COLOR 13,3,3:CLS
140 LINE (75,10)-(175,200),,B
150 PAINT (76,11)
160 CIRCLE (125,60),40,15
170 PAINT (125,60),15
180 LINE (125,60)-(125,25),1
190 LINE (125,60)-(115,80),1
200 LINE (80,120)-(170,190),4,BF
210 LINE (X1,120)-(X2,170),10
220 CIRCLE (X2,170),12,10
230 PAINT (X2,171),10
240 RETURN
```

Ce programme permet de tracer des dessins différents sur la page 0 et sur la page 1 en mode SCREEN 5. Tout d'abord, la page 0 est définie comme page d'affichage, tandis que la page 1 est fixée comme page active (ligne 20).
Ensuite, le sous-programme, à partir de la ligne 130, trace le dessin suivant:



Ensuite, la page d'affichage et la page active sont toutes deux la page 0 (ligne 50), et le dessin suivant est tracé.



Le dessin de la page 1 reste tel qu'il est. La boucle de la ligne 80 à la ligne 120 change ensuite alternativement la page d'affichage de la page 1 à la page 0, ce qui fait que les deux dessins sont affichés alternativement sur l'écran. Cette démarche donne l'impression de mouvement au balancier de l'horloge.

La spécification de la page active a été omise dans les énoncés SET PAGE aux lignes 80 et 100. Quand elle est omise, la spécification antérieure reste en vigueur.

COPIAGE DE DONNEES GRAPHIQUES

- Copiage de graphiques—COPY
- Copiage entre écrans
- Copiage entre l'écran et la mémoire interne
- Copiage entre l'écran et une disquette
- Copiage entre la mémoire et une disquette
- Opérations logiques

COPIAGE DE GRAPHIQUES

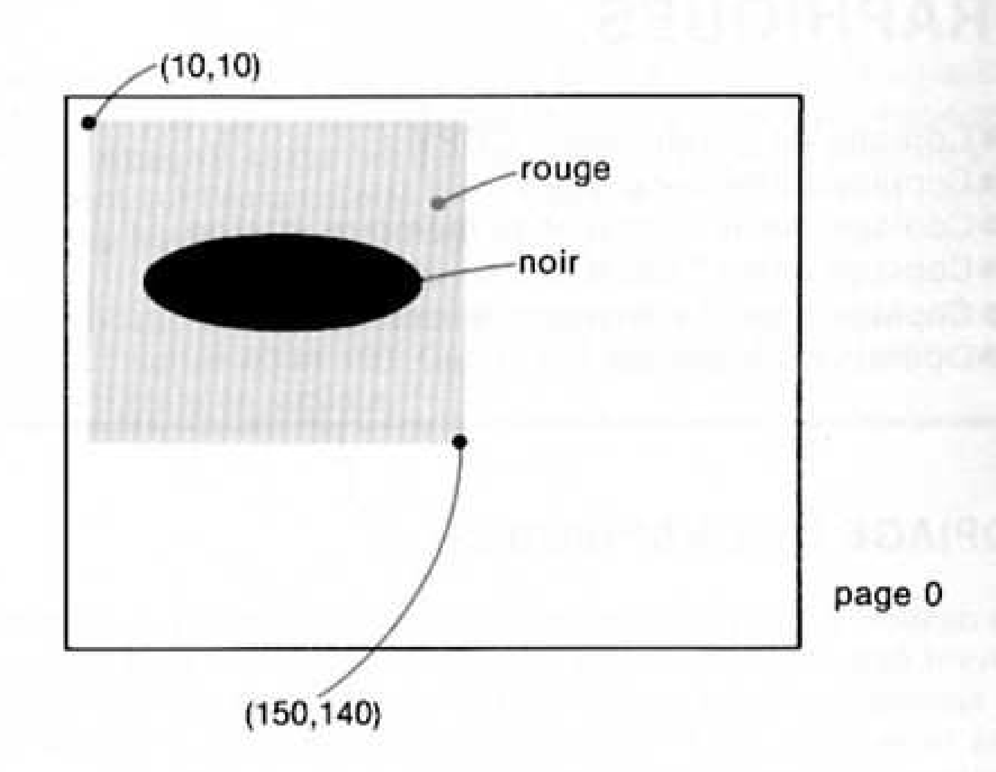
Des dessins tracés en modes graphiques de SCREEN 5 à SCREEN 8 peuvent être copiés. Au cours de ce processus, une zone de l'écran est spécifiée et les données couleur de chaque point, se trouvant dans cette zone, sont copiées en un autre endroit. Il existe trois "endroits" où ces données peuvent être copiées, à savoir:

- l'écran (mémoire vive vidéo ou VRAM)
- une mémoire interne (une variable en tableau)
- une disquette (un fichier)

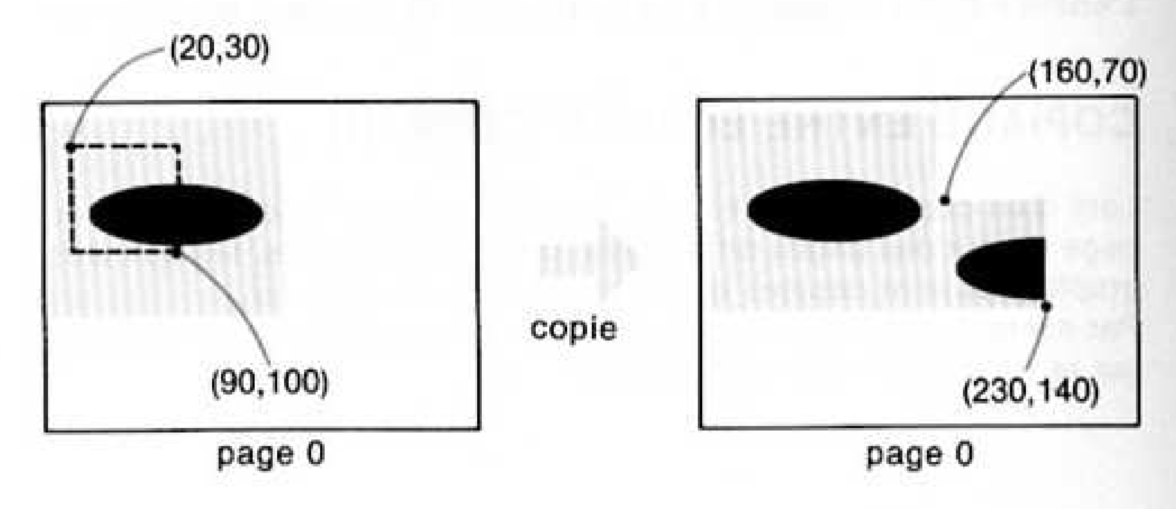
L'énoncé COPY s'emploie pour le copiage de données graphiques.

COPIAGE ENTRE ECRANS COPY (1)

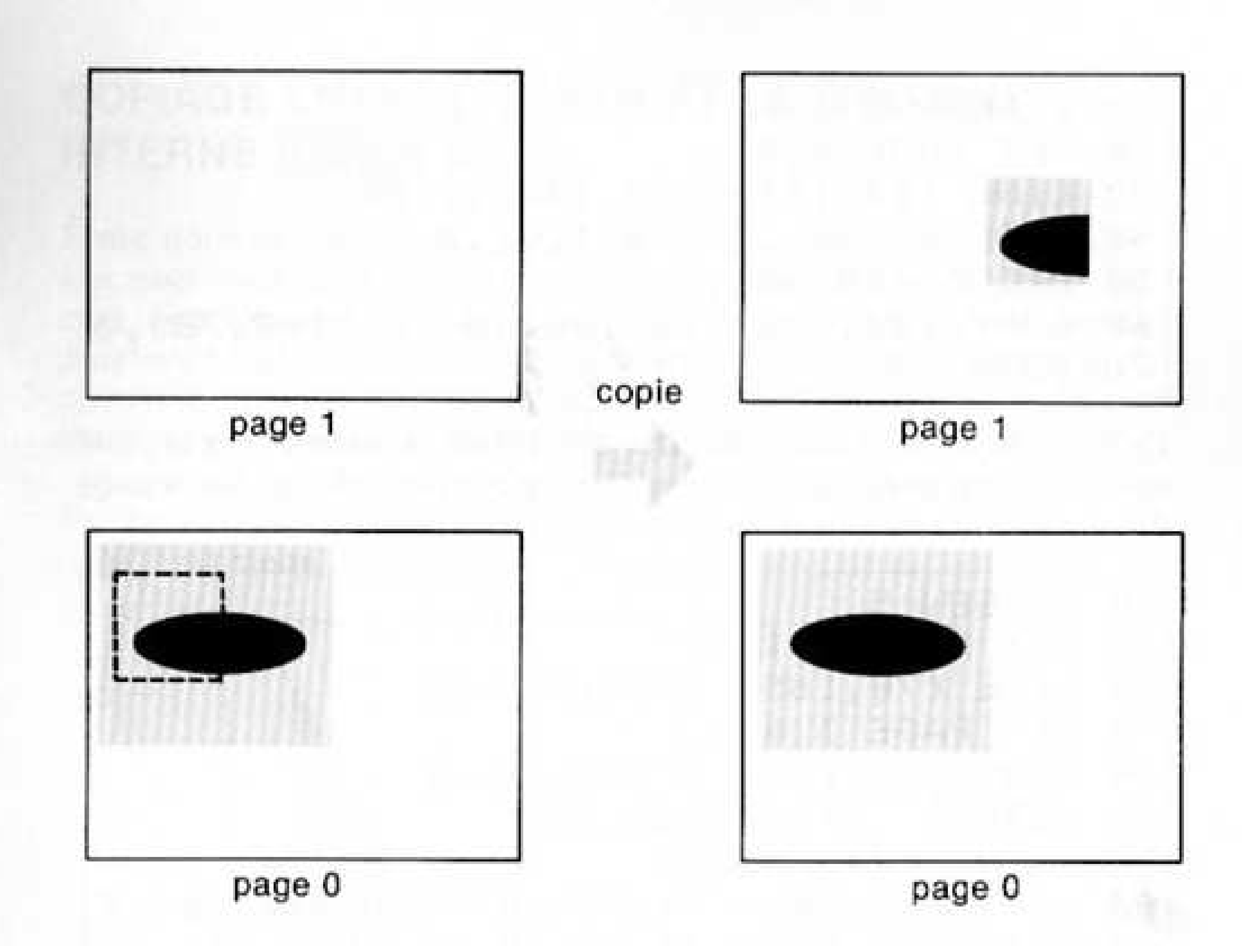
Lors d'un copiage entre écrans, la copie peut se faire sur la même page ou sur des pages différentes. Dans les deux cas, les données graphiques sont copiées dans la VRAM de l'ordinateur.
Par exemple, supposons que les graphiques suivants soient tracés sur la page 0 en mode SCREEN 5.



Si la partie de ce dessin, allant de (20,30) à (90,100) et représentée par les pointillés sur l'illustration, est copiée sur la même page à l'endroit dont (160,70) est la coordonnée du point supérieur gauche, vous obtiendrez le résultat suivant.



Le copiage est également possible sur une page différente. Si vous copiez la même partie du graphique que ci-dessus au même endroit, mais sur la page 1, c'est le résultat suivant que vous obtiendrez.



Lors d'un copiage, les dessins et les couleurs de l'avant-plan sont reproduits à l'endroit de destination indiqué, mais les dessins et couleurs originaux subsistent sur la page de source, l'original.
Le format de l'énoncé COPY est le suivant:

COPY (X1,Y1)—(X2,Y2)[,page source]TO(X3,Y3)
[,page destination]

(X1,Y1) indique la coordonnée du point supérieur gauche de la zone de source, tandis que (X2,Y2) représente la coordonnée du point inférieur droit de la zone de source et que (X3,Y3) définit la coordonnée du point supérieur gauche de la zone de destination. Si la page source et/ou la page destination n'est (ne sont) pas spécifiée(s), la page active sera demandée comme spécification.

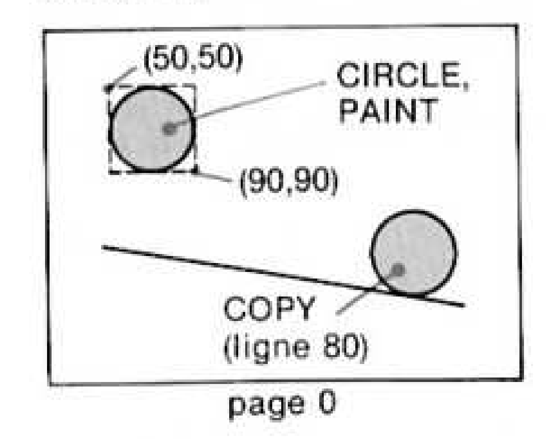
Ceci peut être vérifié en lançant le programme suivant.

```
10 SCREEN 5
20 SET PAGE 0,0
30 LINE (10,10)-(150,140),8,BF
40 CIRCLE (90,60),40,1,,,.3
50 PAINT (90,60),1
60 COPY (20,30)-(90,100),0 TO (160,70),0
70 GOTO 70
```

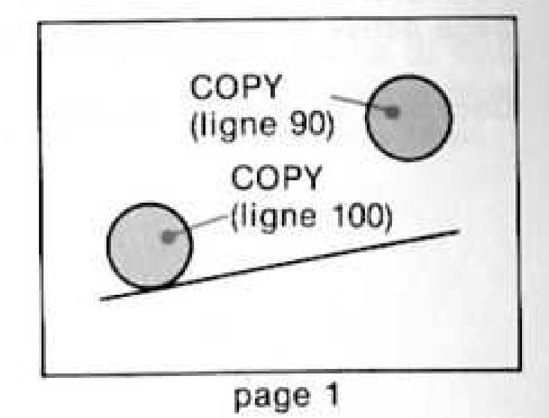
La ligne 60 copie la zone (20,30)—(90,100) de la page 0 vers la zone dont la coordonnée du point supérieur gauche est (160,70) sur la page 0.

```
10 SCREEN 5
20 SET PAGE 0,1:CLS
30 LINE (70,165)-(180,140),1
40 SET PAGE 1,0:CLS
50 LINE (70,140)-(180,165),1
60 CIRCLE (70,70),20,12
70 PAINT (70,70),12
80 COPY (50,50)-(90,90),0 TO (160,120),0
90 COPY (50,50)-(90,90),0 TO (160,50),1
100 COPY (50,50)-(90,90),0 TO (50,120),1
110 SET PAGE 0
120 FOR T=0 TO 300:NEXT T
130 SET PAGE 1
140 FOR T=0 TO 300:NEXT T
150 GOTO 110
```

Ce programme trace les dessins ci-dessous sur la page 0 et la page 1 en mode SCREEN 5. Le premier cercle est tracé sur la page 0 par les énoncés CIRCLE et PAINT des lignes 60 et 70. Les autres cercles sont le résultat du copiage à l'aide de l'énoncé COPY (lignes 80,90,100).



page 0

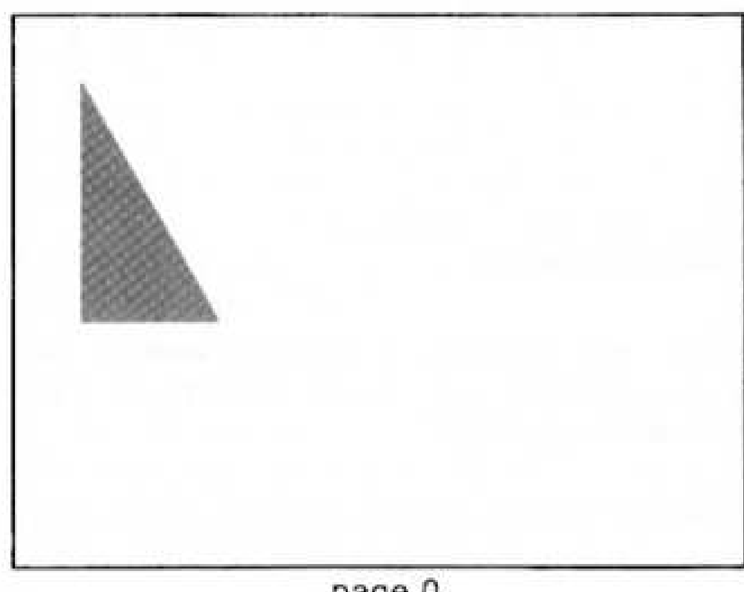


page 1

COPIAGE ENTRE L'ECRAN ET LA MEMOIRE INTERNE COPY (2)

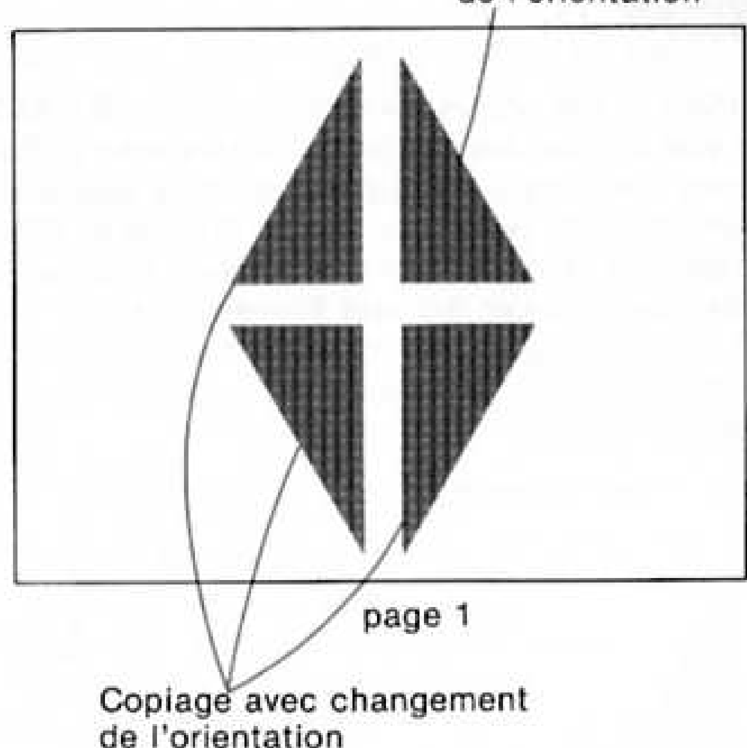
Toute donnée d'écran est conservée dans la mémoire VRAM, mais elle peut aussi être copiée dans la mémoire vive interne (RAM). De plus, des données qui auront été copiées dans la mémoire interne pourront toujours être recopiées sur l'écran (VRAM) et affichées à nouveau. Lorsque des données sont re-copiées vers la mémoire VRAM, leur orientation sur l'écran peut être modifiée.

Par exemple, supposons que vous ayez tracé le dessin suivant sur la page 0 en mode SCREEN 5.



page 0

Si vous copiez ensuite la zone d'écran (20,20)—(60,105) vers la mémoire interne, vous pourrez, par copiage, obtenir l'affichage suivant sur la page 1.



Avant de copier des données graphiques vers la mémoire, vous devrez, pour les recevoir, définir une variable en tableau de type numérique par un énoncé DIM.

Le format d'énoncé COPY ci-après s'emploie pour copier des données graphiques dans la mémoire.

COPY(X1,Y1)—(X2,Y2)[,page source] TO nom de variable en tableau

(X1,Y1) indique la coordonnée du point supérieur gauche et (X2,Y2) celle du point inférieur droit de la zone à copier.
La taille de la variable en tableau est déterminée par la formule suivante:

INT ((((ABS(X1 − X2) + 1) * (ABS(Y1 − Y2) + 1) * taille pixel + 7)/8 + 4)/8) + 1

INT et ABS sont deux fonctions du BASIC qui seront expliquées au Chapitre 7.
La taille d'un pixel (le nombre de pixels, formant un point sur l'écran) varie d'après le mode SCREEN utilisé.

Mode	Taille de pixel
SCREEN 5	4
SCREEN 6	2
SCREEN 7	4
SCREEN 8	8

Par exemple, pour copier les données de la zone (20,20)—(60,105) en mode SCREEN 5, comme X1 est 20 et Y1 est 20 et comme X2 est 60 et Y2 est 105, X1 – X2 sera égal à – 40 et Y1 – Y2 sera égal à – 85. Par conséquent, la taille de la variable en tableau devra être calculée comme suit:

```
INT((((ABS(-40)+1)*(ABS(-85)+1)*4+7)
/8+4)/8)+1
```

Par conséquent, les deux lignes suivantes devront être écrites au début du programme pour définir la variable en tableau P.

```
S=INT((((ABS(-40)+1)*(ABS(-85)+1)
*4+7)/8+4)/8)+1
DIM P(S)
```

Une fois qu'une variable en tableau de ce genre a été définie, l'énoncé COPY peut servir pour copier la donnée de la zone (20,20)—(60,105) dans la mémoire, c'est-à-dire dans la variable en tableau.
Il suffit ensuite de spécifier P dans l'énoncé COPY en procédant comme suit:

```
COPY (20,20)-(60,105),0 TO P
```

L'énoncé COPY ci-dessous s'emploie pour copier les données de la mémoire (de la variable en tableau) vers l'écran (VRAM).

COPY nom de variable en tableau [.orientation] TO(X3,Y3) [.page de destination]

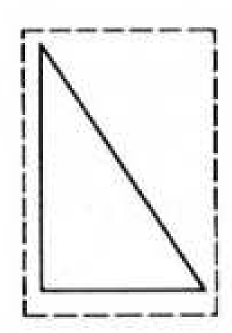
(X3,Y3) est le point de départ pour le tracé des données à copier sur la page de destination.

Il existe quatre "orientations", spécifiées par les numéros de 0 à 3. Par "orientation", on entend la direction dans laquelle le dessin devra être tracé à partir du point de départ.

N° d'orientation	Direction du tracé
0	du haut à gauche vers le bas à droite ↘
1	du haut à droite vers le bas à gauche ↙
2	du bas à gauche vers le haut à droite ↗
3	du bas à droite vers le haut à gauche ↖

0 est demandé lorsque la spécification est omise.

Par exemple, quand les données concernant le graphique suivant ont été copiées dans la mémoire,



La zone entourée d'un pointillé est copiée dans la mémoire.

ces données peuvent être re-copiées comme suit sur l'écran à l'aide des quatre numéros d'orientation et des points de départ du tracé (X3,Y3):

"orientation" 0 "orientation" 1 "orientation" 2 "orientation" 3

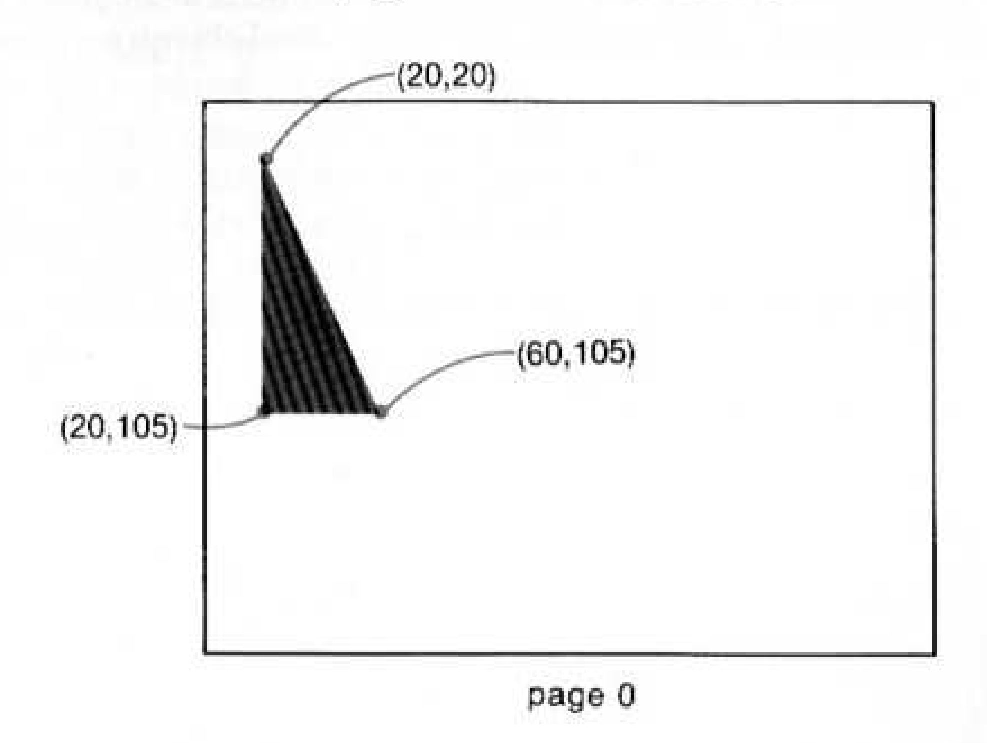
(X3,Y3) (X3,Y3)

(X3,Y3) (X3,Y3)

Si l'on omet le numéro d'orientation, le réglage implicite est 0, c'est-à-dire du haut à gauche vers le bas à droite.
Ecrivons à présent un programme qui affichera l'exemple, donné à la page 132.

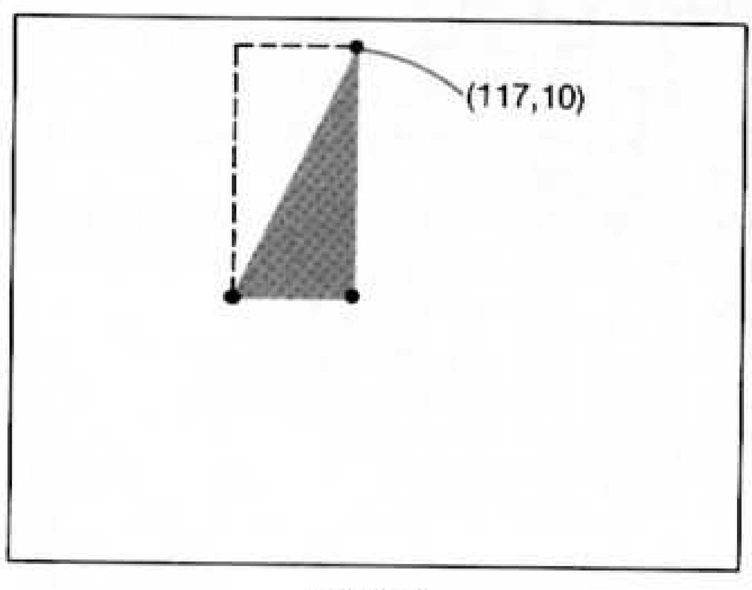
```
10 S=INT((((ABS(-40)+1)*(ABS(-85)+1)*4+7
)/8+4)/8)+1
20 DIM P(S)
30 SCREEN 5
40 SET PAGE 0,1:CLS
50 SET PAGE 0,0:CLS
60 LINE (20,20)-(20,105)
70 LINE (20,20)-(60,105)
80 LINE (20,105)-(60,105)
90 PAINT (25,50)
100 COPY (20,20)-(60,105),0 TO P
110 COPY P,0 TO (138,10),1
120 COPY P,1 TO (117,10),1
130 COPY P,2 TO (138,200),1
140 COPY P,3 TO (117,200),1
150 SET PAGE 1
160 GOTO 160
```

La variable en tableau P est définie par les lignes 10 et 20 et le dessin suivant est tracé sur la page 0 en mode SCREEN 5.



page 0

L'énoncé COPY de la ligne 100 copie donc la zone qui englobe ce dessin, à savoir (20,20)—(60,105) dans la variable en tableau de la mémoire. Ensuite, les lignes 110 à 140 changent l'orientation des données qui ont été copiées dans la mémoire sur la page 1.
Par exemple, la ligne 120 copie les données en utilisant "l'orientation" 1 et (117,10) comme point de départ du dessin.



page 1

Les autres énoncés COPY remplissent des fonctions analogues et un dessin comparable à celui de la page 132 est ainsi obtenu sur la page 1.

COPIAGE ENTRE L'ECRAN ET UNE DISQUETTE COPY (3)

Pourvu que l'ordinateur utilisé soit doté d'une unité de disque interne ou d'une unité de disquette (périphérique), il est également possible de copier vers la disquette, des données qui ont été tracées sur l'écran (VRAM), à l'aide de l'énoncé COPY.

Le format de l'énoncé COPY, destiné au copiage (sauvegarde) de données graphiques sur une disquette, est le suivant:

COPY (X1,Y1)—(X2,Y2) [, page de source] TO "[nom périphérique] ,nom de fichier [.nom de type]"

Les règles, régissant l'affectation d'un nom de fichier, sont exactement les mêmes que celles qui ont servi pour la sauvegarde d'un programme. Le nom de type peut être omis, mais il est conseillé de l'ajouter car il permettra de savoir de quel genre de fichier il s'agit. Dans les exemples de programmes suivants, on utilisera .PIC comme nom de type.
Rédigeons à présent un programme pour copier sur une disquette le même dessin que celui que nous avons placé dans la mémoire.

```
10 SCREEN 5
20 SET PAGE 0,0:CLS
30 LINE (20,20)-(20,105)
40 LINE (20,20)-(60,105)
50 LINE (20,105)-(60,105)
60 PAINT (25,50)
70 COPY (20,20)-(60,105),0 TO "TRIANGLE.
PIC"
```

Un triangle est tout d'abord tracé par les lignes de 30 à 60. Ensuite, la ligne 70 copie, sur la disquette, les données de la zone (20,20)—(60,105) qui englobe le triangle. Donnons TRIANGLE comme nom de fichier et .PIC comme nom de type.
Lorsque le programme est lancé, un triangle est tracé sur la page 0 en mode SCREEN 5. Sitôt après, l'unité de disque entre en service et les données graphiques sont copiées sur la disquette.
Lorsque le copiage est achevé, le programme se termine et le message Ok apparaît sur l'écran.
Si l'instruction FILES est exécutée, le nom de fichier et le nom de type

```
TRIANGLE.PIC
```

sont affichés et vous pouvez ainsi vérifier que les données graphiques sont bel et bien copiées sur la disquette.

Le format de l'énoncé COPY pour le re-copiage de données graphiques de la disquette vers l'écran (VRAM) est le suivant:

COPY "[nom périphérique] nom fichier [. nom type]" [, orientation] TO (X3,Y3) [, page de destination]

Le même nom de fichier et le même nom de type que ceux qui ont été utilisés pour la copie sur disquette doivent être spécifiés. "L'orientation" et les coordonnées du point de départ sont aussi les mêmes que celles, utilisées pour le copiage depuis la mémoire.
Le programme suivant change les données copiées sur le disque selon les quatre orientations différentes et il copie ces données sur l'écran.

```
10 SCREEN 5
20 SET PAGE 1,1:CLS
30 COPY "TRIANGLE.PIC",0 TO (138,10),1
40 COPY "TRIANGLE.PIC",1 TO (117,10),1
50 COPY "TRIANGLE.PIC",2 TO (138,200),1
60 COPY "TRIANGLE.PIC",3 TO (117,200),1
70 GOTO 70
```

Les énoncés COPY des lignes 30 à 60 copient le dessin, sur la page 1 et en mode SCREEN 5, en suivant des points de départ et des orientations différents. Si vous exécutez ce programme, vous verrez comment les données sont copiées.

Dans le programme précédent, la page d'affichage et la page active étaient toutes deux la page 1. Mais si l'on définit des pages différentes comme page active et page d'affichage et que les données de l'image sont tout d'abord copiées de la disquette vers la page active, puis que, après copiage, la page active est changée en page d'affichage, le dessin achevé sera affiché d'un seul coup sur l'écran. Modifiez comme suit le programme précédent, puis lancez-le.

```
10 SCREEN 5
20 SET PAGE 0,1:CLS←——changé
30 COPY "TRIANGLE.PIC",0 TO (138,10),1
40 COPY "TRIANGLE.PIC",1 TO (117,10),1
50 COPY "TRIANGLE.PIC",2 TO (138,200),1
60 COPY "TRIANGLE.PIC",3 TO (117,200),1
65 SET PAGE 1←——ajouté
70 GOTO 70
```

COPIAGE ENTRE LA MEMOIRE ET UNE DISQUETTE COPY (4)

Le format de l'énoncé COPY pour copier, vers une variable en tableau de la mémoire, des données graphiques sauvegardées sur une disquette, est le suivant:

COPY "[nom périphérique] nom de fichier [. nom type]" TO nom de variable en tableau

Pour copier des données graphiques d'une variable en tableau de la mémoire vers une disquette, utiliser le format suivant:

COPY nom de variable en tableau TO "[nom périphérique] nom fichier [. nom type]"

OPERATIONS LOGIQUES

Lors du copiage de données graphiques par un énoncé COPY, il est possible de se livrer à des opérations logiques entre le code couleur utilisé pour la couleur du dessin et le code couleur, défini pour l'écran de destination.
Les dix opérations logiques suivantes peuvent être utilisées avec l'énoncé COPY.

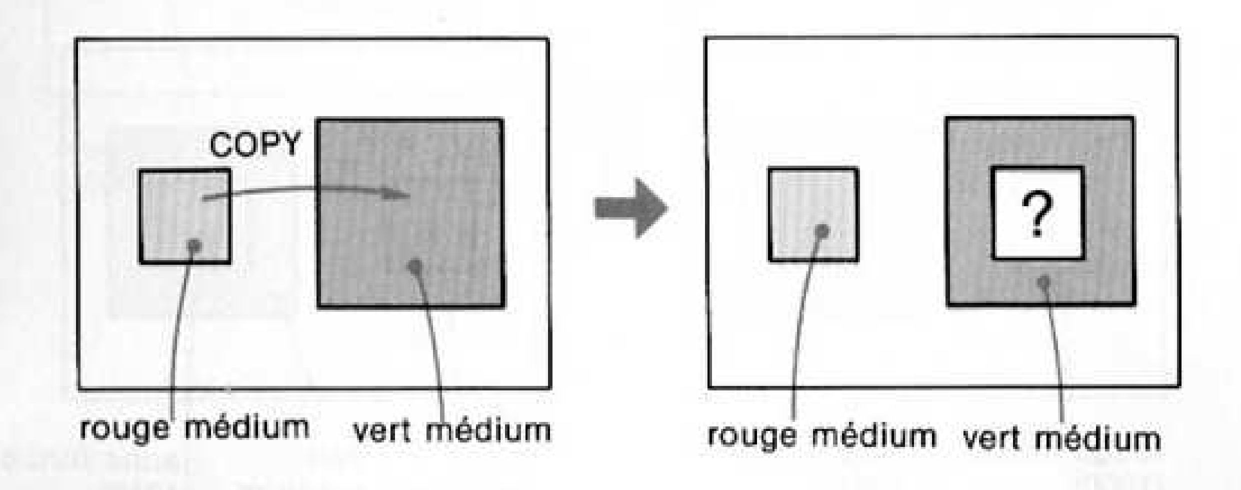
PSET, PRESET, XOR, OR, AND,

TPSET, TPRESET, TXOR, TOR, TAND

Le résultat de l'opération logique est visible quand les codes couleurs sont changés en codes binaires. Le tableau ci-après indique les codes binaires des 16 couleurs en mode SCREEN 5 lors de la mise en service du BASIC.

Couleur	Code couleur (décimal)	Code couleur (binaire)	Couleur	Code couleur (décimal)	Code couleur (binaire)
transparent	0	0000	rouge médium	8	1000
noir	1	0001	rouge clair	9	1001
vert médium	2	0010	jaune foncé	10	1010
vert clair	3	0011	jaune clair	11	1011
bleu foncé	4	0100	vert foncé	12	1100
bleu clair	5	0101	magenta	13	1101
rouge foncé	6	0110	gris	14	1110
bleu ciel	7	0111	blanc	15	1111

Observons ce qui se passe quand une figure rouge médium (code couleur 1000) est copiée sur une figure vert médium (code couleur 0010).



Si aucune opération logique n'est effectuée, le carré rouge médium sera superposé au carré vert médium. Cependant, si l'on effectue une opération logique, la couleur de la portion [?] du carré vert médium sera une couleur autre que le rouge médium. La couleur du carré devient alors tributaire de l'opération logique qui est effectuée.

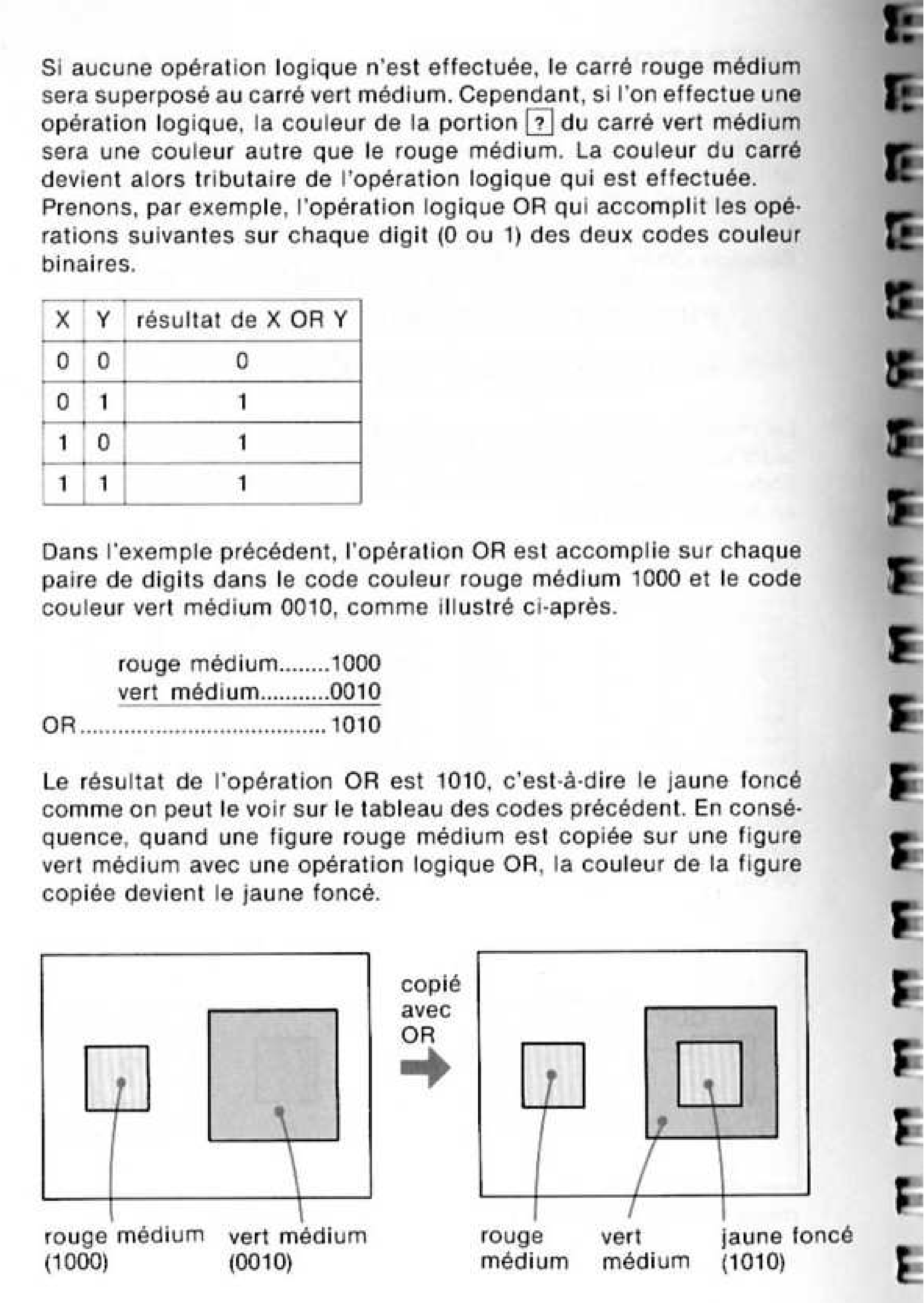
Prenons, par exemple, l'opération logique OR qui accomplit les opérations suivantes sur chaque digit (0 ou 1) des deux codes couleur binaires.

X	Y	résultat de X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

Dans l'exemple précédent, l'opération OR est accomplie sur chaque paire de digits dans le code couleur rouge médium 1000 et le code couleur vert médium 0010, comme illustré ci-après.

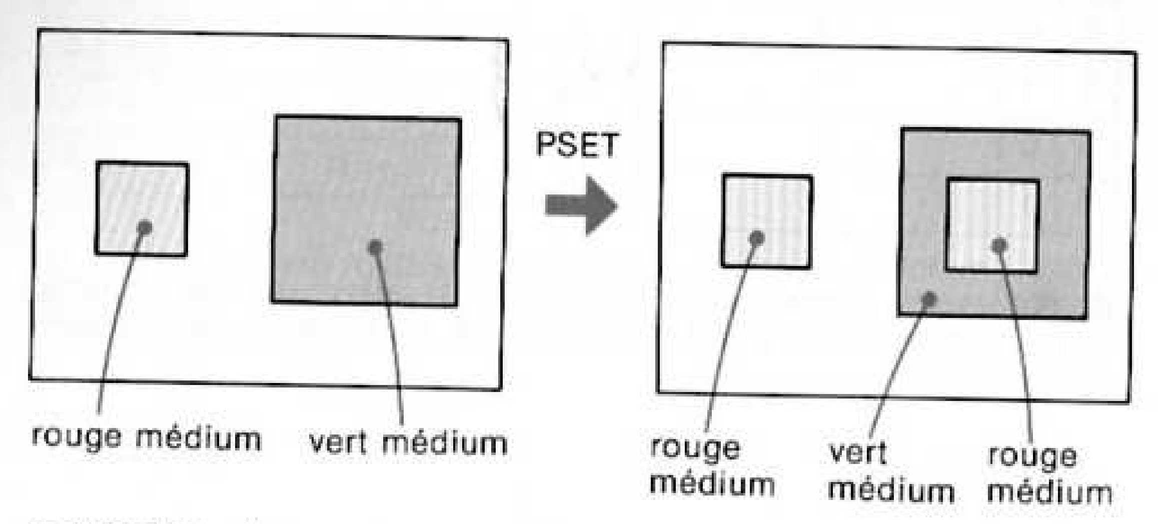
rouge médium........1000
vert médium...........0010
OR...1010

Le résultat de l'opération OR est 1010, c'est-à-dire le jaune foncé comme on peut le voir sur le tableau des codes précédent. En conséquence, quand une figure rouge médium est copiée sur une figure vert médium avec une opération logique OR, la couleur de la figure copiée devient le jaune foncé.

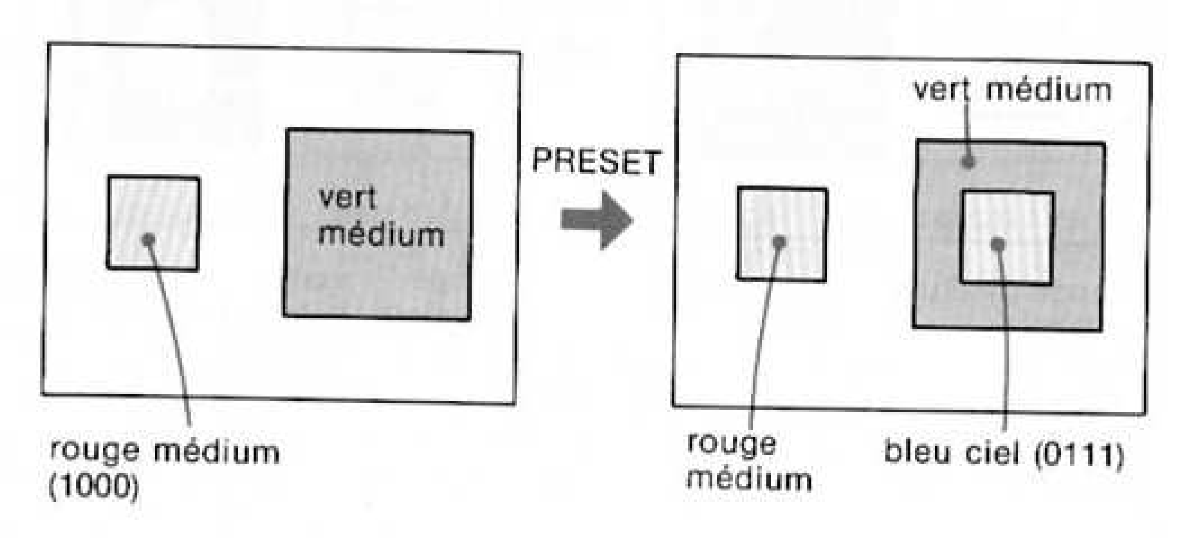


On trouvera ci-après les résultats obtenus en se servant des opérations logiques, autres que OR.

- PSET—la couleur copiée reste la même, quelle que soit la couleur de la zone de destination.



- PRESET—change chaque digit du code couleur de la couleur copiée en son digit contraire, c.à.d. 0 en 1 et 1 en 0, sans aucun rapport avec la couleur de la zone de destination. Par exemple, si le rouge médium (1000) est copié par PRESET, la couleur deviendra le bleu ciel (0111).

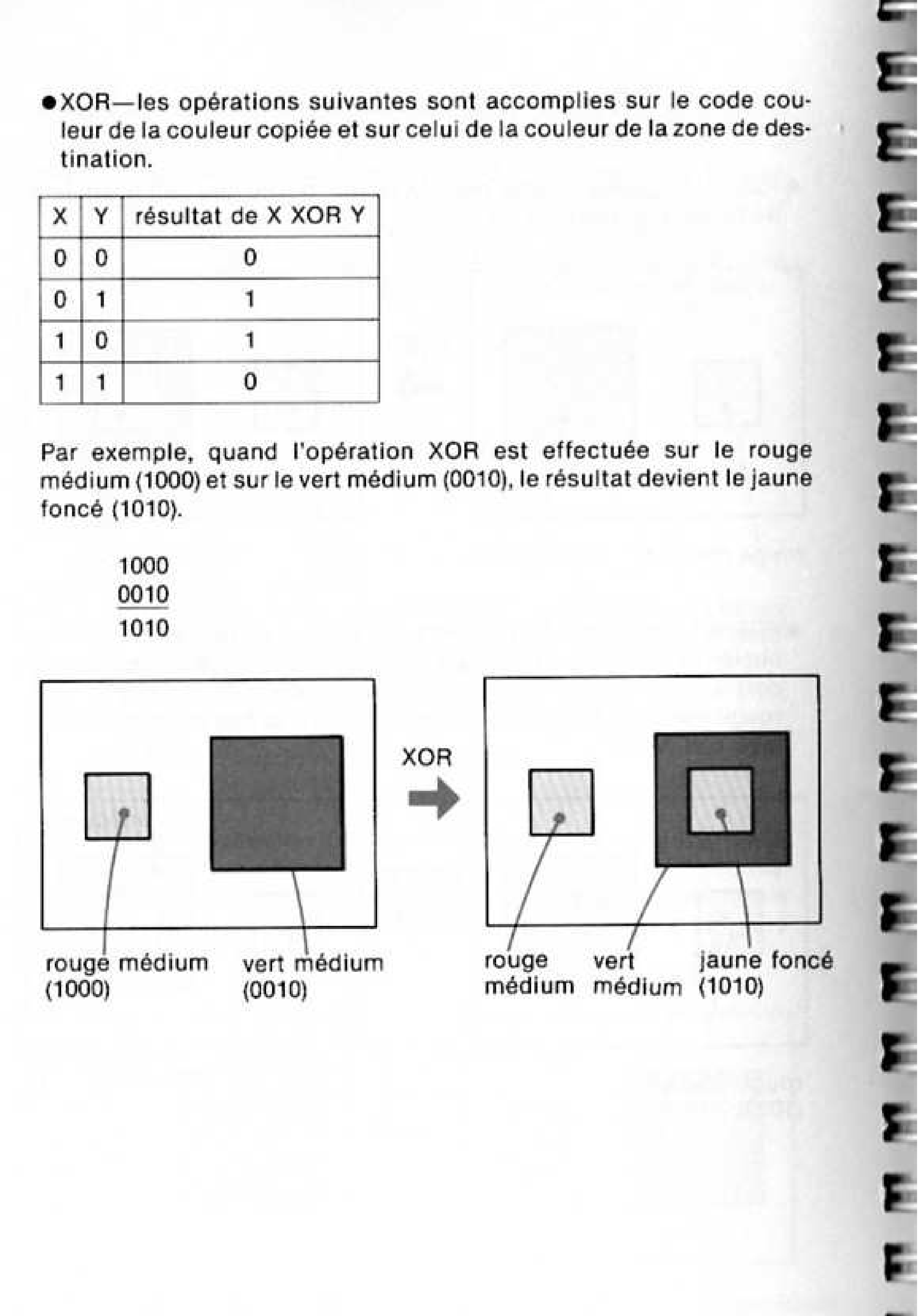


●XOR—les opérations suivantes sont accomplies sur le code couleur de la couleur copiée et sur celui de la couleur de la zone de destination.

X	Y	résultat de X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Par exemple, quand l'opération XOR est effectuée sur le rouge médium (1000) et sur le vert médium (0010), le résultat devient le jaune foncé (1010).

1000
0010
1010

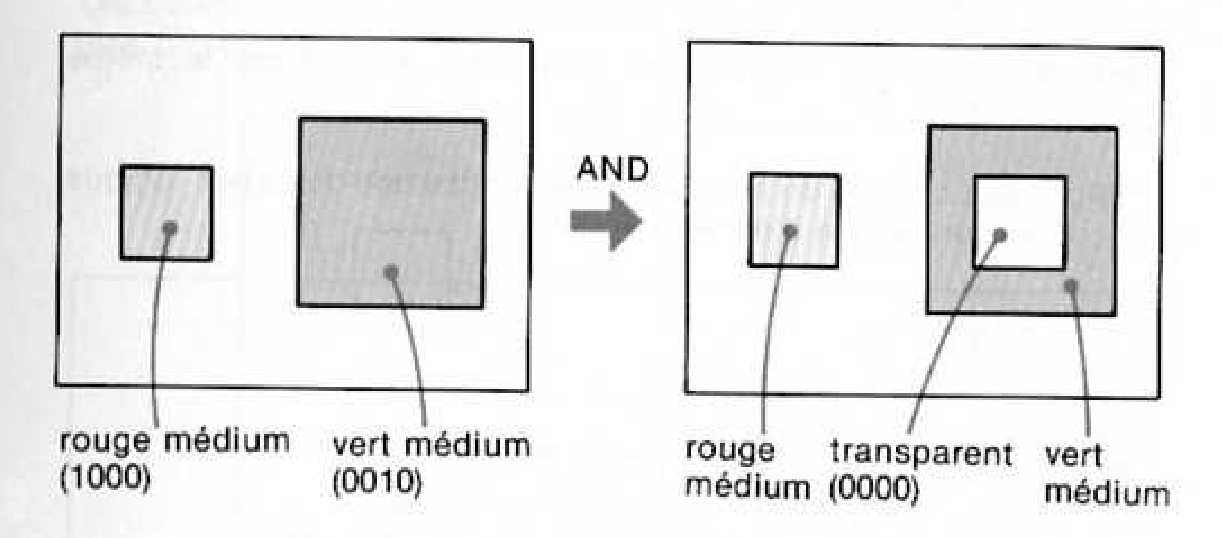


● AND—effectue les opérations suivantes sur le code couleur de la couleur copiée et sur celui de la zone de destination.

X	Y	résultat de X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

Quand l'opération AND est effectuée sur le rouge médium (1000) et sur le vert médium (0010), le résultat devient le transparent (0000).

1000
0010
0000



● TPSET, TPRESET, TOR, TXOR, TAND—ces opérations sont identiques à celles de PSET, PRESET, OR, XOR et AND, sauf que quand une couleur est le transparent et que T est utilisé comme préfixe, cette couleur restera le transparent, quelle que soit la couleur de la zone de destination.

Utilisation des opérations logiques avec un énoncé COPY
Les formats de l'énoncé COPY ci-après s'emploient quand des opérations logiques sont effectuées.

● **D'écran vers écran**

COPY (X1,Y1)—(X2,Y2) [, page source] TO (X3,Y3) [, page destination] [, **opération logique**]

Exemple: COPY (10,10)—(100,100), 0 TO (30,30), 1, XOR

● **De mémoire (variable en tableau) vers écran**

COPY nom variable en tableau [, orientation] TO (X3,Y3) [, page destination] [, **opération logique**]

Exemple: COPY P, 1 TO (30,30), 0, TAND

● **De disquette (fichier) vers écran**

COPY "[nom périphérique] nom fichier [. nom type]" [, orientation] TO (X3,Y3) [, page destination] [, **opération logique**]

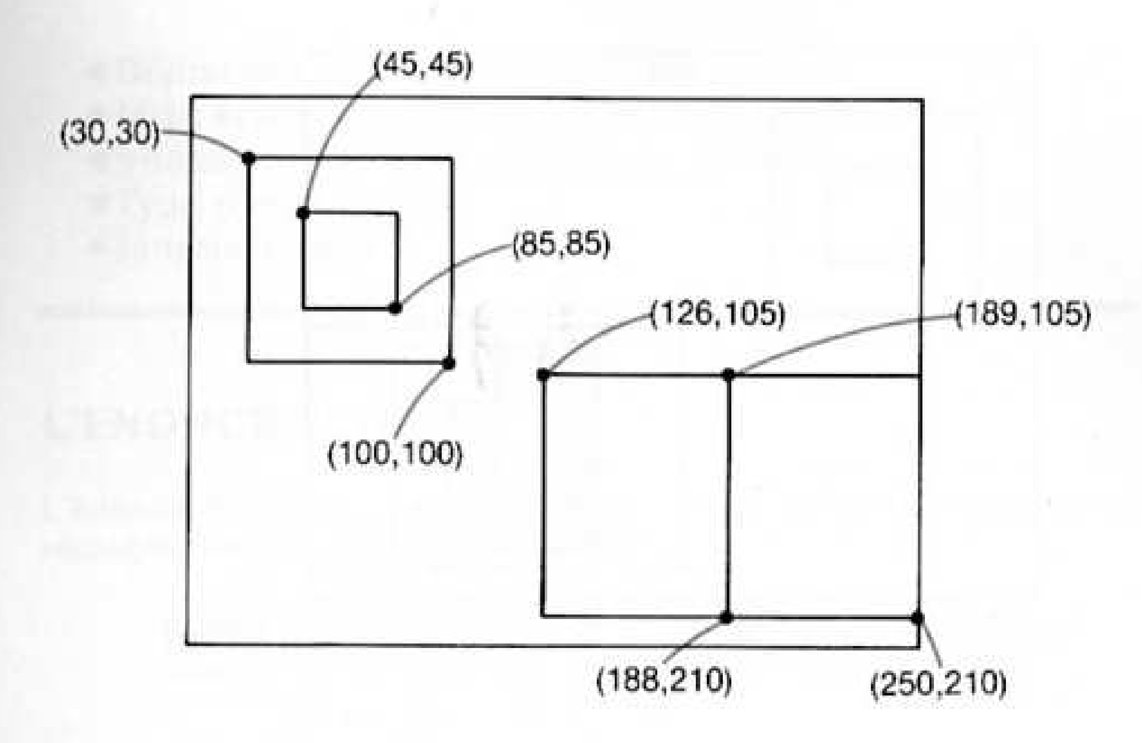
Exemple: COPY "TRIANGLE .PIC", 2 TO (30,30), 1, PRESET

Lorsque l'opération logique est omise, le résultat est le même qu'avec PSET.

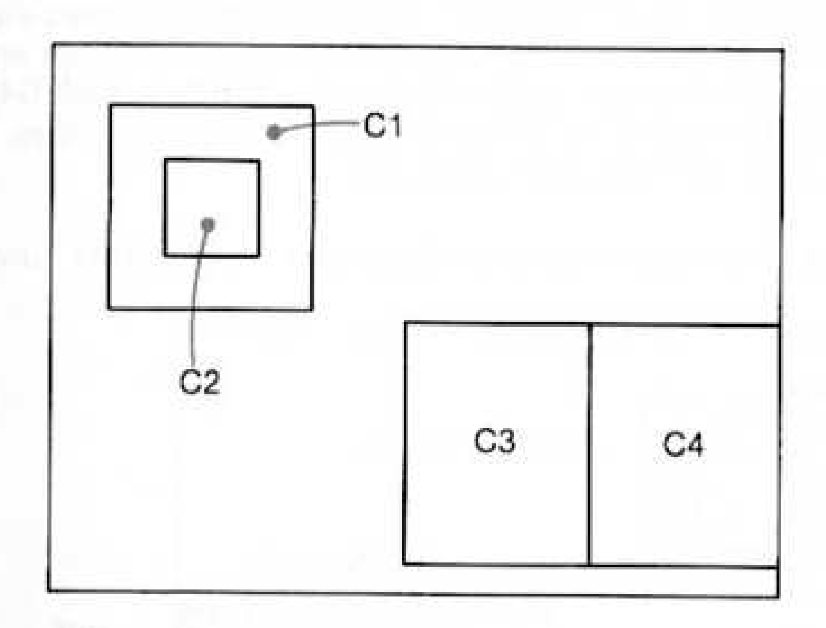
Rédigeons à présent un programme qui utilise une opération logique en copiant un dessin d'écran à écran.

```
10 SCREEN 5
20 SET PAGE 0,0:CLS
30 READ C1,C2,C3,C4
40 LINE (30,30)-(100,100),C1,BF
50 LINE (45,45)-(85,85),C2,BF
60 LINE (126,105)-(188,210),C3,BF
70 LINE (189,105)-(250,210),C4,BF
80 COPY (30,30)-(100,100),0 TO (153,124)
,0,OR
90 GOTO 90
100 DATA 8,3,10,2
```

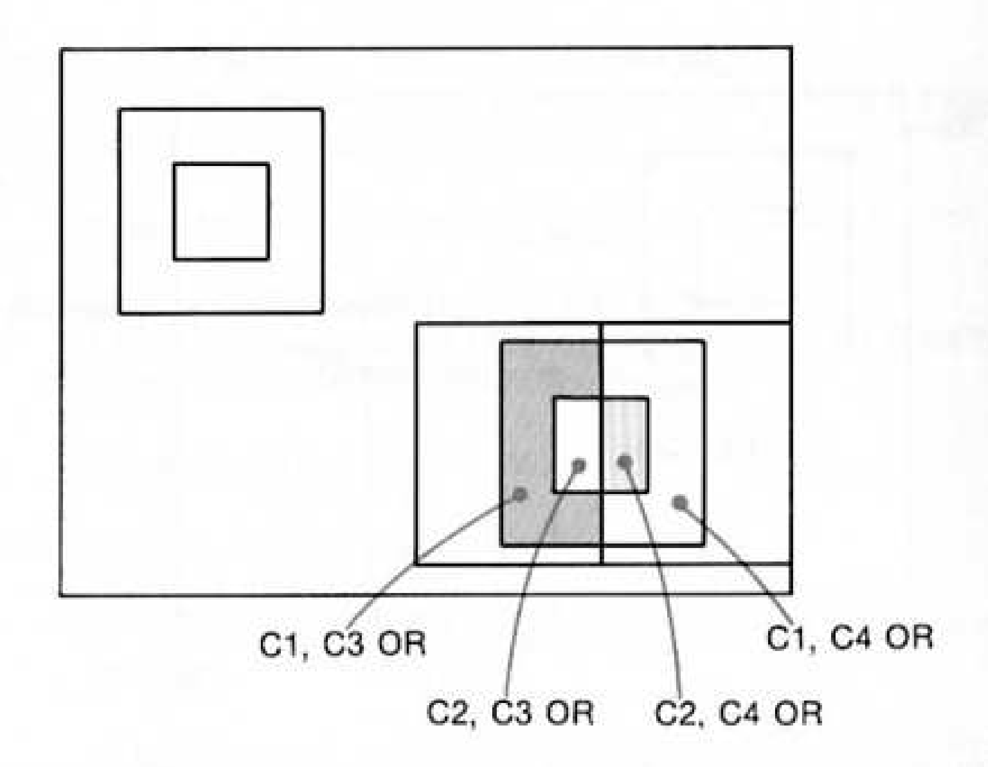
Ce programme réalise le dessin suivant.



Les couleurs seront déterminées par les valeurs affectées aux variables C1, C2, C3 et C4.

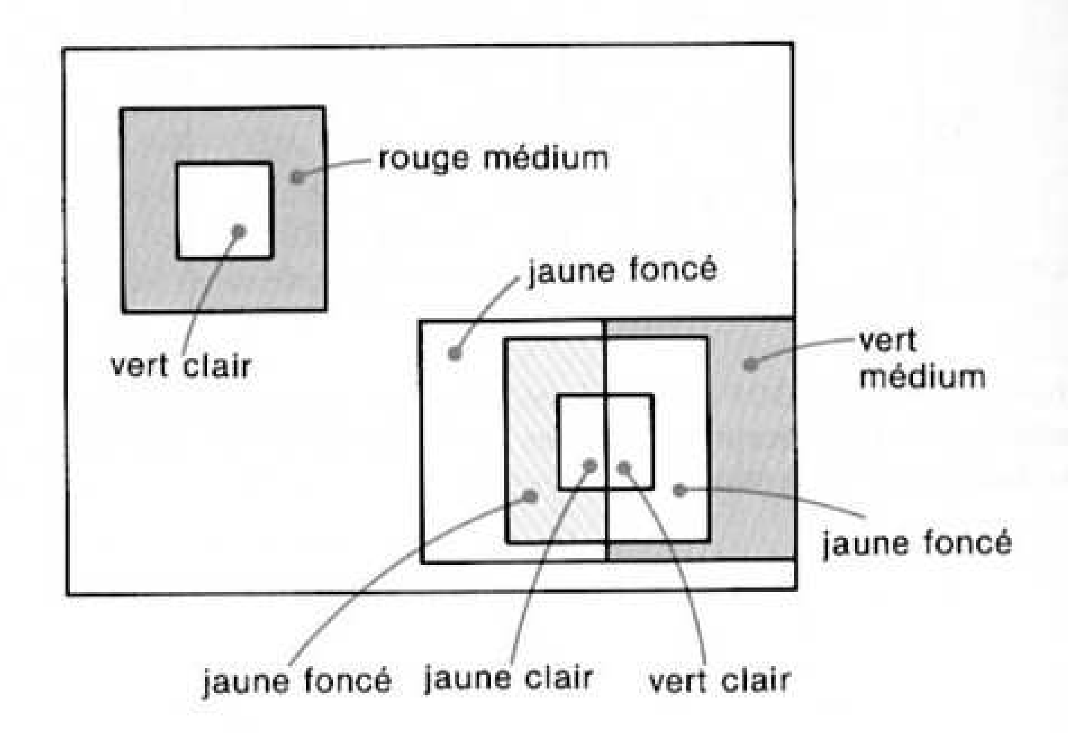


Ensuite, l'énoncé COPY de la ligne 80 copie le dessin. Au cours de cette copie, les opérations logiques sont accomplies sur les portions du dessin qui sont superposées.



Ce programme montre les résultats des opérations logiques qui combinent quatre couleurs différentes. Les couleurs, affectées aux diverses variables dans le programme, sont: le rouge médium en C1, le vert clair en C2, le jaune foncé en C3 et le vert médium en C4 (ce qui correspond à l'énoncé DATA de la ligne 100). Dans ce cas, OR est l'opération logique effectuée (ligne 80).

En lançant ce programme, vous obtiendrez les résultats suivants:



L'ENONCE SCREEN

- ●Définitions de l'énoncé SCREEN
- ●Mise en/hors service du déclic des touches
- ●Vitesse de transfert d'interface cassette
- ●Type d'imprimante
- ●Entrelacement

L'ENONCE SCREEN

L'énoncé SCREEN s'emploie pour effectuer un certain nombre de réglages, en plus du mode SCREEN.

SCREEN [mode], [taille sprite], [déclic touches], [vitesse transfert], [type imprimante], [entrelacement]

COMMANDE DE DECLIC DES TOUCHES, VITESSE DE TRANSFERT, TYPE D'IMPRIMANTE

Mise en/hors service du déclic des touches

Le troisième paramètre (déclic touches) de l'énoncé SCREEN permet de spécifier si une tonalité sera, ou non, audible à chaque poussée sur les touches du clavier. Aucun son ne sera produit si la commande est réglée sur 0. Tout réglage à un nombre différent (de 1 à 255) produira une tonalité en réponse à une action sur une touche.

```
SCREEN ,,0 —— pas de tonalité
SCREEN ,,1 —— tonalité audible
```

Réglage de la vitesse de transfert sur cassette

Cette vitesse (ou taux en bauds) indique le nombre de bits de données, transmis en une seconde. Un taux en bauds de 1200 signifie donc que 1200 bits de données sont transmis par seconde, tandis qu'un taux en bauds de 2400 marque une vitesse de transfert de 2400 bits de données par seconde.
Le quatrième paramètre de l'énoncé SCREEN spécifie le taux en bauds à utiliser pour la transmission des données vers/depuis le magnétocassette. Le taux en bauds 1 équivaut à 1200 bauds, tandis que le taux 2 équivaut à 2400 bauds. La valeur implicite initiale est 1 (soit 1200 bauds).

```
SCREEN ,,,1 —— 1200 bauds
SCREEN ,,,2 —— 2400 bauds
```

Si le taux en bauds a été réglé sur 2400 par l'énoncé SCREEN ,,,2 avant la sauvegarde d'un programme sur une cassette, ce même taux de 2400 bauds devra être défini par l'énoncé SCREEN ,,,2 avant de charger le même programme depuis la cassette.

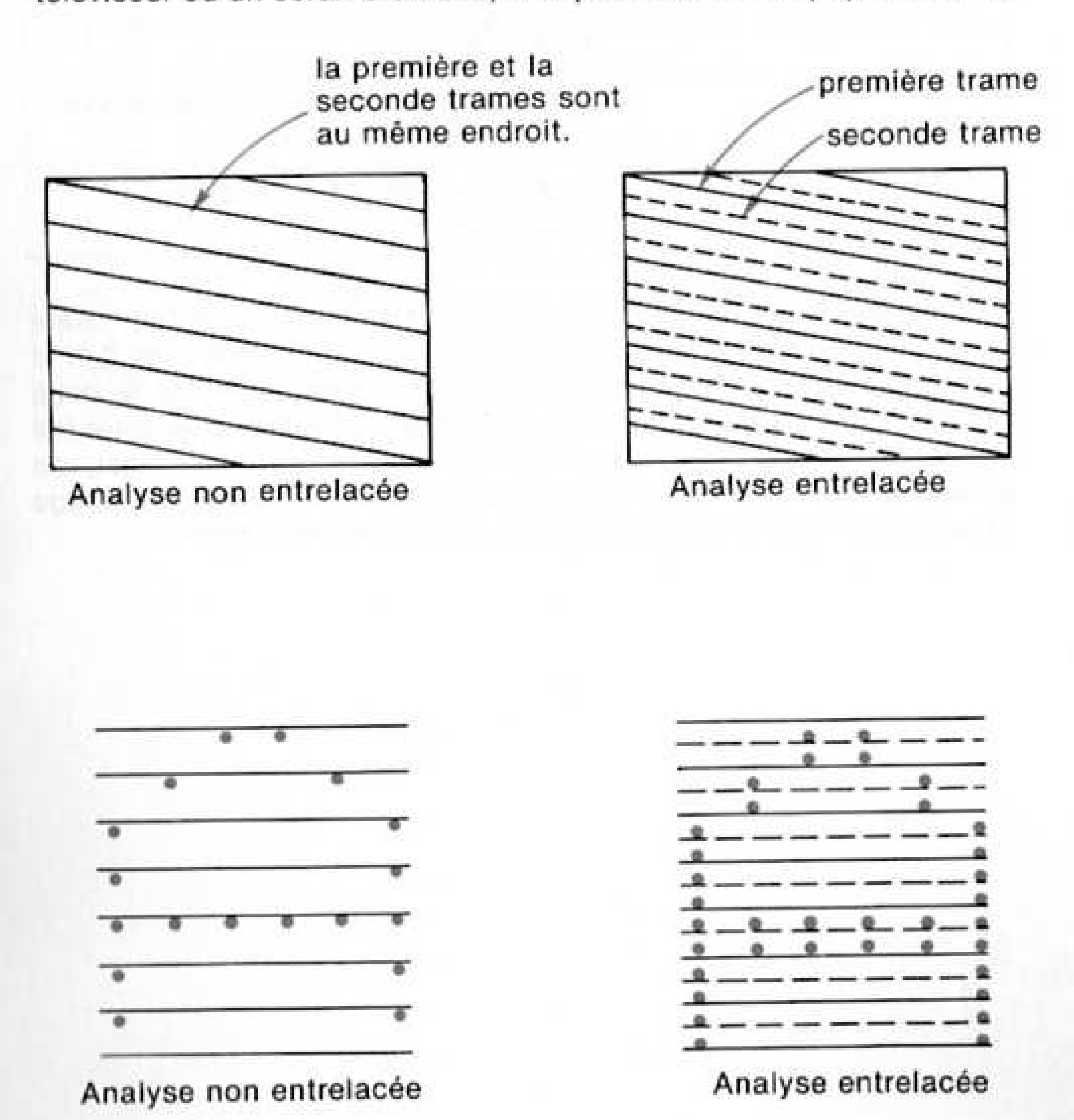
Type d'imprimante

Le cinquième paramètre de l'énoncé SCREEN fait en sorte que le réglage soit conforme au type d'imprimante utilisé. 0 est le réglage pour les imprimantes de type MSX. Tout réglage autre que 0 (de 1 à 255) convient pour les autres types d'imprimante. La valeur implicite initiale est 0 (imprimante de type MSX).
Les imprimantes de type MSX sont conçues spécialement en fonction des ordinateurs MSX et elles disposent d'une police de caractères graphiques MSX spéciaux.
Si l'on fait appel à une imprimante qui n'est pas du type MSX et que l'énoncé SCREEN ,,,,1 est exécuté, les caractères graphiques spéciaux du système MSX seront imprimés comme espaces vierges.

LE MODE ENTRELACEMENT

Analyse entrelacée
Normalement, les ordinateurs MSX accomplissent une analyse non entrelacée. Cependant, celle-ci peut être changée en analyse entrelacée (ou analyse à intercalage) grâce au sixième paramètre (mode entrelacement) de l'énoncé SCREEN.

Quand on fait appel à l'analyse entrelacée, les emplacements de la première trame analysée et de la deuxième trame sont différents, ce qui fournit un affichage plus détaillé. Si l'on désire utiliser l'analyse entrelacée, il faudra faire appel à un écran moniteur disposant de propriétés de longue persistance phosphorescente; faute de quoi, sur un téléviseur ou un écran ordinaire, il se produira un fort papillotement.



Le mode d'affichage de pages alternées paires/impaires

Le paramètre d'entrelacement de l'énoncé SCREEN peut servir pour choisir le mode d'affichage de pages alternées paires/impaires. Pour choisir ce mode, la page d'affichage doit d'abord être déterminée comme page impaire (1 ou 3) avec l'énoncé PAGE. Ensuite, quand le mode d'affichage de pages alternées paires/impaires est spécifié par l'énoncé SCREEN, la page d'affichage et la page, numérotée d'une unité de moins que la page d'affichage, seront affichées alternativement à une cadence rapide.

Le tableau ci-dessous indique les réglages du mode entrelacement.

Mode entrelacement	Mode spécifié
0	normal (non entrelacé, pas de pages alternées)
1	mode entrelacé
2	mode d'affichage de pages alternées paires/impaires
3	pages alternées paires/impaires, mode entrelacement

Le programme suivant trace un cercle jaune sur la page 0 et un ovale blanc sur la page 1 en mode SCREEN 5. Tout d'abord, la page 0 et la page 1 sont affichées alternativement par changement de la page d'affichage, à l'aide de l'énoncé SET PAGE, et l'intervalle entre les changements de page est progressivement raccourci. Ainsi, on pourra obtenir un effet intéressant en passant au mode d'affichage de pages alternées paires/impaires à la fin du programme.

```
10 COLOR 15,4,4:SCREEN 5,,,,,0
20 SET PAGE 0,1:CLS
30 XC=128 : YC=100
40 '*** draw yellow circle ***
50 SET PAGE 0,0
60 CIRCLE (XC,YC),45,10
70 PAINT (XC,YC),10
80 FOR T=0 TO 2000:NEXT T
90 '*** draw white oval ***
100 SET PAGE 1,1
110 CIRCLE (XC,YC),90,15,,,.7
120 PAINT (XC,YC),15
130 FOR T=0 TO 2000:NEXT T
140 '*** change pages ***
150 J=1200:DP=0:AP=1
160 SET PAGE DP,DP:SWAP DP,AP
170 FOR T=0 TO J:NEXT T
180 J=J*.8:IF J>1 THEN 160
190 '*** even/odd mode ***
200 SET PAGE 1,1
210 SCREEN ,,,,,2
220 GOTO 220
```

Chapitre 6
Les motifs sprite ou les lutins

DEFINITION ET UTILISATION DES MOTIFS SPRITE

- Les motifs sprite ou les lutins
- Définition des motifs sprite—SPRITE$
- Affichage d'un motif sprite—PUT SPRITE
- Animation de motifs sprite

LES MOTIFS SPRITE OU LES LUTINS

Un "lutin", appelé également "sprite", est un motif défini librement et composé de 8×8 points ou de 16×16 points, qui peut être déplacé sur l'écran. En MSX2-BASIC, ces motifs peuvent être affichés et déplacés sur 32 "plans sprite" différents.

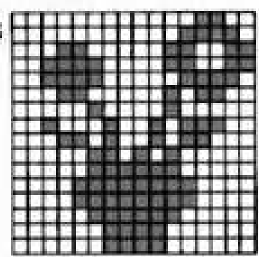
Le mode écran et les motifs sprite
Les motifs sprite peuvent s'employer en modes SCREEN de 1 à 8, c'est-à-dire tous à l'exception de SCREEN 0. Un motif sprite, utilisé dans les modes SCREEN de 4 à 8, possède diverses fonctions qui ne sont pas disponibles en modes SCREEN de 1 à 3.

Les types de motifs sprite

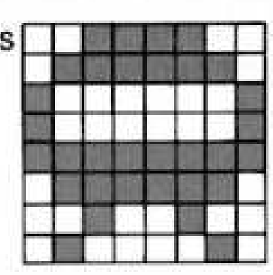
Un motif sprite ou "lutin" se compose soit de 8×8 points, soit de 16×16 points. Chacun peut être affiché soit en taille normale, soit en taille agrandie, cette dernière étant le double de la normale, tant dans le sens vertical qu'horizontal.

8×8 points normal

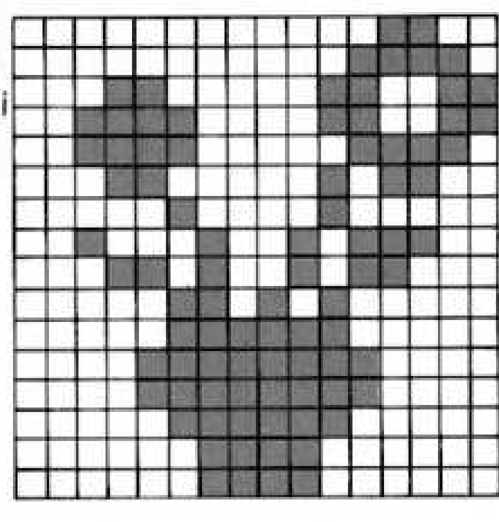
16×16 points normal



8×8 points agrandi



16×16 points agrandi



Spécification de la taille de sprite—énoncé SCREEN

Le second paramètre de l'énoncé SCREEN choisit la taille du motif sprite.

Mode SCREEN, taille de sprite

Taille de sprite	Taille choisie
0	8×8 points, normal
1	8×8 points, agrandi
2	16×16 points, normal
3	16×16 points, agrandi

Par exemple,

```
SCREEN 2,3
```

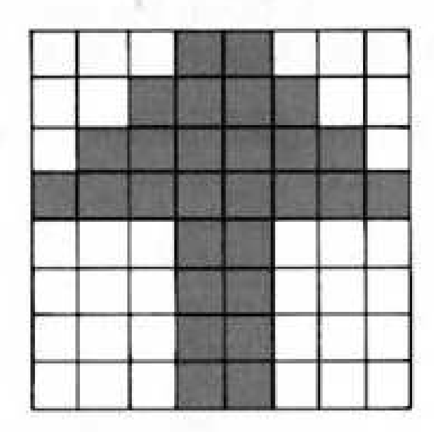
spécifiera le mode SCREEN 2 et choisira la taille de sprite 16×16 points, agrandi. Une fois que la taille de sprite est choisie par l'énoncé SCREEN, les lutins de tous les plans sprite seront affichés dans cette même taille.

DEFINITION DES MOTIFS SPRITE

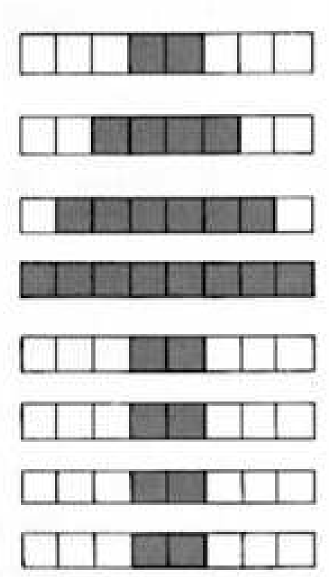
SPRITE$ variable

Le motif sprite à 8×8 points

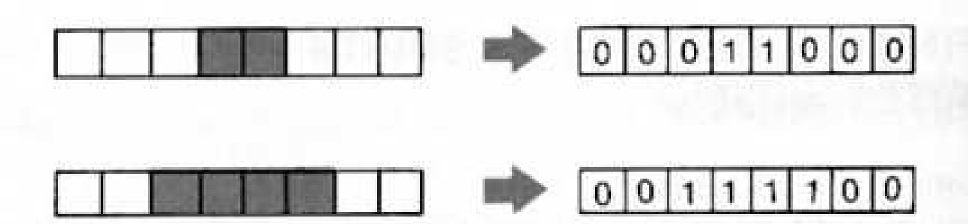
Lorsqu'un motif de 8×8 points est défini, il est tout d'abord séparé en 8 lignes horizontales. Ainsi, par exemple, le motif d'une flèche est défini comme sur la figure suivante.



Chacune des 8 lignes horizontales est divisée en un petit motif qui comprend 8 points.



Ensuite, un 1 est affecté à un point marqué, tandis qu'un 0 l'est à un point non marqué, de manière à obtenir une notation binaire. Par exemple, la ligne supérieure sera exprimée par 00011000 et la seconde le sera par 00111100.

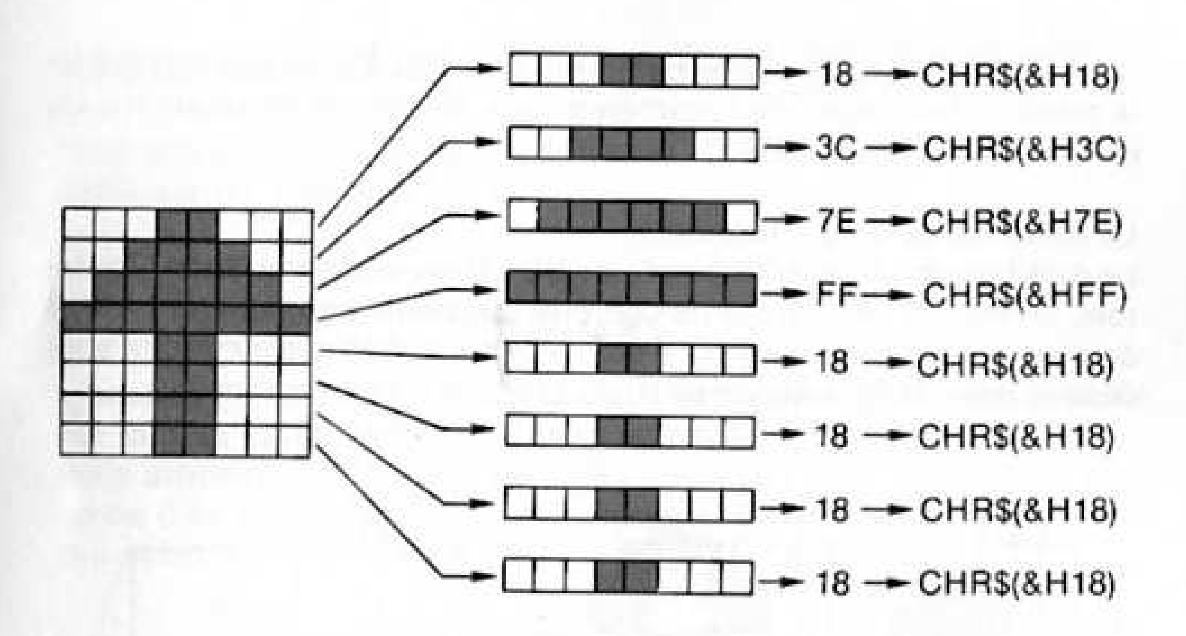


Les nombres binaires sont alors convertis en notation hexadécimale ou décimale. La ligne supérieure devient alors 00011000 (binaire) = 18 (hexadécimal) ou 24 (décimal). Quant à la seconde ligne, elle devient 00111100 (binaire) = 3C (hexadécimal) ou 60 (décimal).

Le tableau suivant s'emploie pour convertir des nombres binaires en hexadécimaux.

Motif	Hexadécimal	Motif	Hexadécimal
□□□□	0	■□□□	8
□□□■	1	■□□■	9
□□■□	2	■□■□	A
□□■■	3	■□■■	B
□■□□	4	■■□□	C
□■□■	5	■■□■	D
□■■□	6	■■■□	E
□■■■	7	■■■■	F

Tout d'abord, le motif en 8 points est divisé en 4 points sur le côté gauche et 4 points sur le côté droit. Ensuite, le tableau précédent est utilisé pour déterminer les équivalents hexadécimaux.
Pour le motif □□□■■□□□, les 4 points de gauche □□□■ correspondent à la valeur hexadécimale 1 sur le tableau précédent, tandis que les 4 points de droite ■□□□ correspondent à 8. Par conséquent, l'équivalent hexadécimal doit être 18.
La démarche suivante consiste à obtenir le caractère pour lequel la valeur hexadécimale (ou décimale) est le code de caractère en utilisant la fonction CHR$, comme indiqué ci-dessous.



La donnée ainsi obtenue par le motif sprite de 8 × 8 points est ajoutée de façon séquentielle depuis le haut, et elle est affectée à la variable SPRITE$ pour définir le motif sprite.
Le motif de la flèche de l'exemple précédent sera défini comme suit:

```
SPRITE$(1)=CHR$(&H18)+CHR$(&H3C)+CHR$(&H
7E)+CHR$(&HFF)+CHR$(&H18)+CHR$(&H18)+CHR
$(&H18)+CHR$(&H18)
```

Le numéro du motif sprite ainsi défini est 1, comme l'indique le numéro 1, placé entre parenthèses dans SPRITE$(1).
(L'emploi des fonctions sera expliqué au Chapitre 7.)

SPRITE$ (numéro sprite) = chaîne de caractères

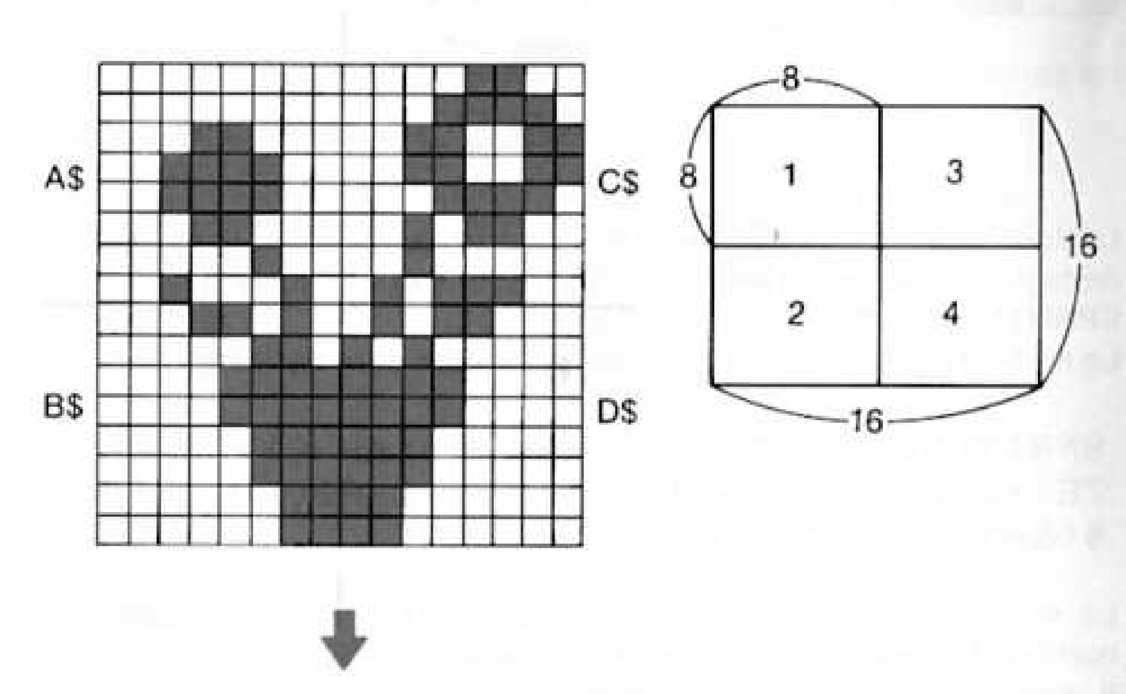
De plus, s'il existe un caractère qui peut être obtenu avec une fonction CHR$, il pourra être utilisé directement dans une chaîne de caractères. Dans l'exemple précédent, comme CHR$(&H3C) est "<" et que CHR$(&H7E) est "~", ces deux caractères peuvent être utilisés directement comme suit:

```
SPRITE$(1)=CHR$(&H18)+"<"+"~"+CHR$(&HFF)
+CHR$(&H18)+CHR$(&H18)+CHR$(&H18)+CHR$(&
H18)
```

(En ce qui concerne la conversion en caractères, prière de consulter le tableau des codes de caractères dans le manuel de référence de programmation en BASIC.)

Le motif sprite à 16×16 points

Le motif sprite de 16×16 points se définit de la même façon. Toutefois, un motif à 16×16 points doit être considéré comme étant formé de quatre motifs sprite de 8×8 points chacun. Ces quatre motifs sont définis dans l'ordre suivant:




```
A$=CHR$(&H0)+CHR$(&H0)+CHR$(&H18)+CHR$(&
H3C)+CHR$(&H3C)+CHR$(&H18)+CHR$(&H4)+CHR
$(&H22)
B$=CHR$(&H1A)+CHR$(&H6)+CHR$(&HF)+CHR$(&
HF)+CHR$(&H7)+CHR$(&H7)+CHR$(&H3)+CHR$(&
H3)
C$=CHR$(&HC)+CHR$(&H1E)+CHR$(&H33)+CHR$(
33)+CHR$(&H1E)+CHR$(&H2C)+CHR$(&H20)+CHR
$(&H5C)
D$=CHR$(&H58)+CHR$(&HA0)+CHR$(&HF0)+CHR$
(&HF0)+CHR$(&HE0)+CHR$(&HE0)+CHR$(&HC0)+
CHR$(&HC0)
SPRITE$(2)=A$+B$+C$+D$
```

Nombre de motifs sprite pouvant être définis

Un maximum de 256 motifs sprite de 8×8 points peuvent être définis à l'aide des nombres de 0 à 255, tandis qu'un maximum de 64 motifs sprite de 16×16 points peuvent l'être, par les nombres de 0 à 63.

AFFICHAGE D'UN MOTIF SPRITE PUT SPRITE

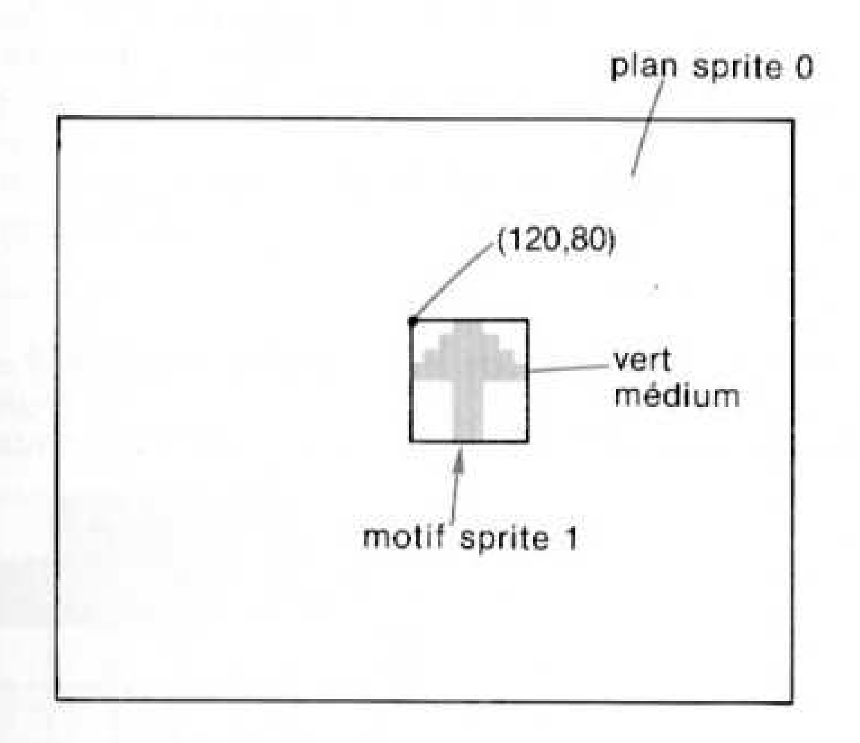
Un énoncé PUT SPRITE est utilisé pour afficher un motif sprite déterminé sur un plan sprite.

PUT SPRITE numéro de plan sprite, [(X,Y)], [couleur], [numéro motif sprite]

L'énoncé PUT SPRITE affiche le motif sprite avec le numéro spécifié sur le plan sprite spécifié et dans la couleur spécifiée.
Pour afficher le motif sprite défini ci-dessus (motif N°1) sur le plan sprite 0 en vert médium (code couleur 2) à la position (120,80), il faudra exécuter:

```
PUT SPRITE 0,(120,80),2,1
```

L'emplacement d'affichage spécifié correspond au point du coin supérieur gauche dans le cadre du motif sprite. Les coordonnées X,Y sont spécifiées dans un système de coordonnées sur l'écran graphique qui considère (0, – 1) comme origine (0,0).



Règles d'affichage des motifs sprite (pour les modes SCREEN 1, 2, 3)

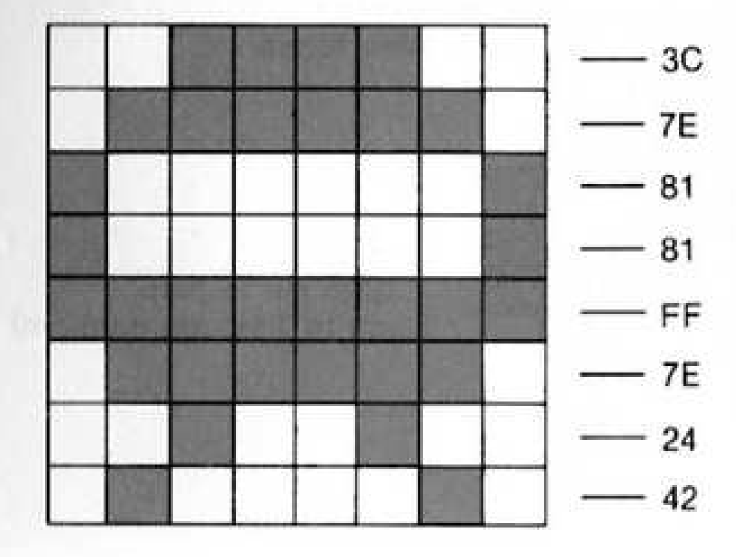
- Un seul motif sprite peut être affiché par plan sprite.
- Lorsque des motifs sprite chevauchent sur des plans sprite différents, le motif du plan arrière (celui dont le numéro est le plus grand) est caché par celui du plan avant.
- Lorsque cinq motifs sprite ou davantage sont arrangés en ligne horizontale, les quatre premiers (ceux dont les numéros de plans sprite sont plus petits) sont affichés.
- Lorsque l'emplacement d'affichage est omis, celui qui a été spécifié par la dernière instruction graphique est considéré comme l'emplacement spécifié.
- Lorsque le code couleur est omis, la couleur de l'avant-plan est considérée comme la couleur spécifiée.
- Lorsque le numéro de motif sprite est omis, le numéro du plan sprite est considéré comme le numéro spécifié.

ANIMATION DE MOTIFS SPRITE

Un motif sprite ou "lutin" est animé en exécutant de façon répétée l'énoncé PUT SPRITE tout en changeant l'emplacement d'affichage spécifié dans l'énoncé. A chaque exécution de l'énoncé PUT SPRITE, le "lutin" précédent dans le même plan sprite disparaît, de sorte qu'il n'est pas nécessaire d'effacer chaque fois ce lutin dans le programme. De plus, comme l'emplacement d'affichage est changé par unités de 1 point, le mouvement du lutin se fait en douceur et sans saccade.
Dans le programme suivant, un motif "lutin" en forme d'OVNI se déplace en diagonale sur l'écran.

```
10 SCREEN 2,0
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 COLOR ,1,1:CLS
40 X=120:Y=50:VX=1:VY=1
50 PUT SPRITE 0,(X,Y),5,0
60 X=X+VX
70 IF X>240 OR X<0 THEN VX=-VX
80 Y=Y+VY
90 IF Y>180 OR Y<0 THEN VY=-VY
100 GOTO 50
```

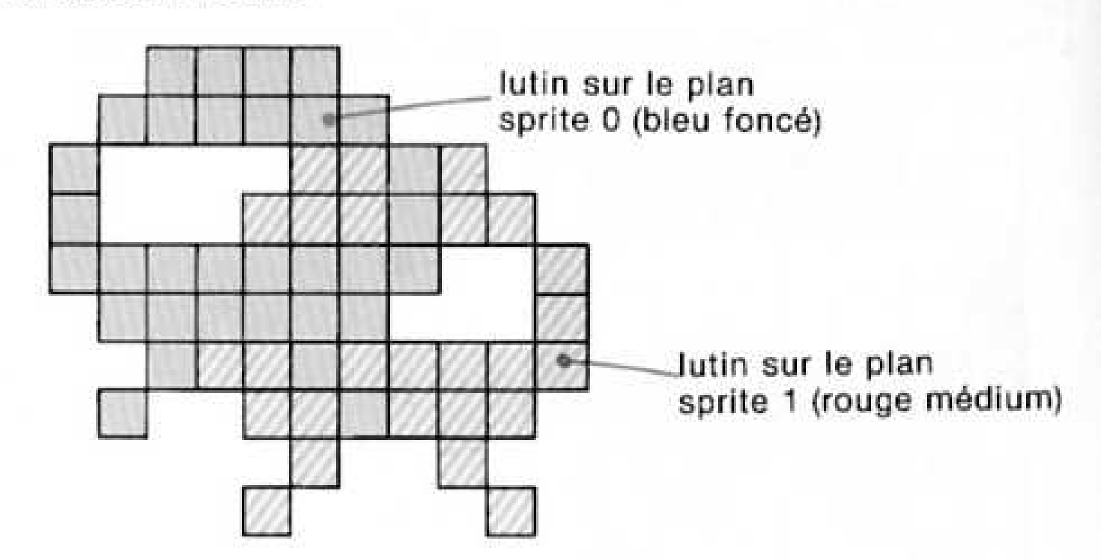
Le mode SCREEN utilisé est 2 et le sprite a la taille normale à 8×8 points (ligne 10).
L'illustration suivante présente le "lutin", défini à la ligne 20.



Ce motif sprite est affiché par l'énoncé PUT SPRITE de la ligne 50. Les valeurs initiales de (X,Y), qui indiquent l'emplacement d'affichage, sont fixées à (120,50). Ensuite, les valeurs (X,Y) sont changées aux lignes 60 à 80 et le programme repasse à la ligne 50. De cette façon, le lutin se déplace sur l'écran.
Le programme suivant fait comprendre ce qui se passe quand se chevauchent deux motifs sprite.

```
10 SCREEN 2,1
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 COLOR ,1,1:CLS
40 FOR X=0 TO 117
50 PUT SPRITE 0,(X,80),4,0
60 PUT SPRITE 1,(240-X,84),8,0
70 NEXT X
80 GOTO 80
```

Le motif sprite agrandi de 8 × 8 points a été utilisé de manière à mieux visualiser le chevauchement. Le lutin est le même que celui qui a servi dans le programme précédent. Ce motif est affiché sur les plans sprite 0 et 1 par les deux énoncés PUT SPRITE des lignes 50 et 60. Le motif sur le plan sprite 0 est bleu foncé et il se déplace de la gauche vers la droite. Par contre, le motif du plan 1 est de couleur rouge médium et il se déplace de la droite vers la gauche. Lorsque les deux lutins se chevauchent, celui du plan 0 (bleu foncé) apparaît comme étant devant l'autre.



UTILISATION DES FONCTIONS SPRITE AMELIOREES

- Les fonctions sprite améliorées
- Changement de la couleur d'un sprite—COLOR SPRITE
- Spécification de la couleur de chaque ligne d'un sprite—COLOR SPRITE$
- Technique de définition d'un sprite

LES FONCTIONS SPRITE AMELIOREES

Des sprites aux fonctions améliorées sont utilisables en modes SCREEN 4 à SCREEN 8.
Le tableau suivant illustre les améliorations des fonctions sprite dans les modes SCREEN 4 à 8, en comparaison des modes SCREEN 1 à 3.

Fonction	SCREEN 1—3	SCREEN 4—8
Nombre de sprites affichés sur une ligne horizontale	Un maximum de 4	Un maximum de 8
Couleur de sprite	1 couleur pour 1 sprite	Un maximum de 8 couleurs (8×8 points), ou 16 couleurs (16×16 points) pour 1 sprite
Changement de couleur de sprite	Spécifié par l'énoncé PUT SPRITE	Spécifié par l'énoncé PUT SPRITE ou l'énoncé COLOR SPRITE

CHANGEMENT DE LA COULEUR D'UN SPRITE COLOR SPRITE

La couleur d'un sprite est spécifiée par l'énoncé PUT SPRITE, mais dans les modes SCREEN 4 à 8, cette couleur peut être changée par un énoncé COLOR SPRITE après qu'elle a été affichée.

COLOR SPRITE (numéro plan sprite) = numéro de palette

L'énoncé COLOR SPRITE change la couleur du lutin dans le plan spécifié et selon la couleur spécifiée.

Par exemple, pour changer en rouge médium (code couleur 8) la couleur du lutin, affiché au plan sprite 2, il faudra exécuter:

```
COLOR SPRITE(2)=8
```

Quand le programme suivant est exécuté, 6 OVNI de couleurs différentes atterrissent, puis leur couleur est changée. La dernière couleur à laquelle ils sont changés est le transparent et, à l'étape suivante, leur atterrissage se répète.

```
10 SCREEN 5,1
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 COLOR ,1,1:CLS
40 X1=0:Y1=118
50 FOR L=1 TO 14
60 READ X2,Y2
70 LINE (X1,Y1)-(X2,Y2),3
80 X1=X2:Y1=Y2
90 NEXT L
100 PAINT (1,119),3
110 P=0:X=5:YE=101:C=3:GOSUB 250
120 P=1:X=45:YE=122:C=4:GOSUB 250
130 P=2:X=83:YE=87:C=5:GOSUB 250
140 P=3:X=133:YE=143:C=6:GOSUB 250
150 P=4:X=168:YE=127:C=7:GOSUB 250
160 P=5:X=218:YE=96:C=8:GOSUB 250
170 FOR S=1 TO 10
180 FOR SP=0 TO 5
190 SC=S+3:IF S=10 THEN SC=0
200 COLOR SPRITE (SP)=SC
210 NEXT SP
220 FOR T=0 TO 300:NEXT T
230 NEXT S
240 GOTO 110
250 FOR Y=-8 TO YE
260 PUT SPRITE P,(X,Y),C,0
270 NEXT Y
280 RETURN
290 DATA 26,118,40,139,67,139
300 DATA 79,104,103,104,128,160
310 DATA 154,160,161,144,192,144
320 DATA 200,113,252,113,252,212
330 DATA 0,212,0,118
```

trace l'arrière-plan (lignes 40–100)

déplace les lutins (lignes 110–160)

change la couleur des lutins (lignes 170–230)

sous-programme de mouvement des lutins (lignes 250–280)

données pour le tracé de l'arrière-plan (lignes 290–330)

L'énoncé COLOR SPRITE de la ligne 200 change la couleur des lutins. Les numéros des plans sprite sont affectés à la variable SP. Le code couleur (la variable SC) change de 3 à 13, mais à la dernière répétition de la boucle (S = 10), la couleur devient 0 (le transparent) et les OVNI sont rendus invisibles.

SPECIFICATION DE LA COULEUR DE CHAQUE LIGNE D'UN SPRITE COLOR SPRITE$

Comme vous le savez déjà, un sprite de 8×8 points se compose de huit lignes horizontales, tandis qu'un de 16×16 points en comprend seize horizontales. En modes SCREEN de 4 à 8, il est possible de spécifier séparément la couleur de chacune de ces lignes.

COLOR SPRITE$ (numéro de plan sprite) = chaîne de caractères

L'énoncé COLOR SPRITE$ fait appel à une chaîne de caractères pour spécifier la couleur de chaque ligne d'un lutin sur le plan sprite spécifié.
La chaîne de caractères dans l'énoncé COLOR SPRITE$ est composée de

CHR$ (code couleur)

ces chaînes de caractères étant réunies par le signe +. Le premier CHR$ (code couleur) spécifie la couleur de la première ligne, le second CHR$ (code couleur) celle de la seconde ligne, et ainsi de suite.

— code couleur de ligne 1 ... 1
— code couleur de ligne 2 ... 2
— code couleur de ligne 3 ... 3
— code couleur de ligne 4 ... 4
— code couleur de ligne 5 ... 5
— code couleur de ligne 6 ... 6
— code couleur de ligne 7 ... 7
— code couleur de ligne 8 ... 8

L'énoncé COLOR SPRITE$ suivant spécifiera les couleurs, indiquées dans l'illustration ci-avant, pour un lutin placé sur le plan sprite 0.

```
COLOR SPRITE$(0)=CHR$(1)+CHR$(2)+CHR$(3)
+CHR$(4)+CHR$(5)+CHR$(6)+CHR$(7)+CHR$(8)
```

Si seulement sept chaînes de caractères CHR$ (code couleur) ou moins sont spécifiées, la couleur des lignes non spécifiées ne sera pas changée.
Dans l'énoncé

```
COLOR SPRITE(0)=CHR$(1)+CHR$(2)
```

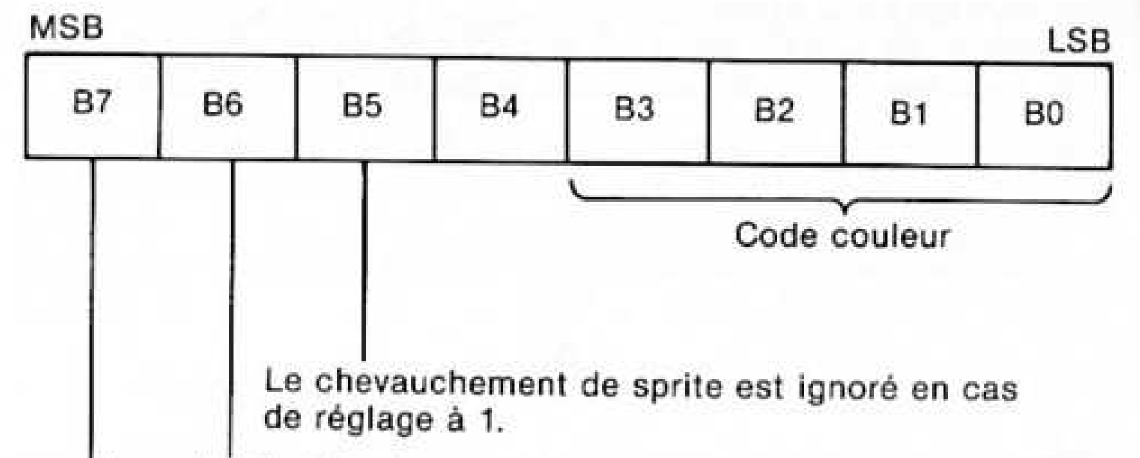
la couleur de la ligne 1 deviendra le code couleur 1 et la couleur de la ligne 2 sera le code couleur 2, mais la couleur des lignes 3 et suivantes restera inchangée.
Le programme suivant constitue une révision du précédent et il permet d'afficher des OVNI aux lignes rouge médium et bleu foncé.

```
10 SCREEN 5,1
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 COLOR ,1,1:CLS
40 X1=0:Y1=118
50 FOR L=1 TO 14
60 READ X2,Y2
70 LINE (X1,Y1)-(X2,Y2),3
80 X1=X2:Y1=Y2
90 NEXT L
100 PAINT (1,119),3
110 P=0:X=5:YE=101:C=3:GOSUB 250
120 P=1:X=45:YE=122:C=4:GOSUB 250
130 P=2:X=83:YE=87:C=5:GOSUB 250
140 P=3:X=133:YE=143:C=6:GOSUB 250
150 P=4:X=168:YE=127:C=7:GOSUB 250
160 P=5:X=218:YE=96:C=8:GOSUB 250
170 FOR SP=0 TO 5
180 COLOR SPRITE$(SP)=CHR$(8)+CHR$(4)+CH
R$(8)+CHR$(4)+CHR$(8)+CHR$(4)+CHR$(8)+CH      ─ changé
R$(4)
190 NEXT SP
200 GOTO 200
250 FOR Y=-8 TO YE
260 PUT SPRITE P,(X,Y),C,0
270 NEXT Y
280 RETURN
290 DATA 26,118,40,139,67,139
300 DATA 79,104,103,104,128,160
310 DATA 154,160,161,144,192,144
320 DATA 200,113,252,113,252,212
330 DATA 0,212,0,118
```

La ligne 180, qui se trouve dans la partie modifiée du programme, spécifie les rayures rouges et bleues de l'OVNI.

Donnée en chaîne de caractères

Les données, utilisables avec la fonction CHR$ dans l'énoncé COLOR SPRITE$, ne sont pas limitées aux codes de couleur. Lorsque la donnée est exprimée en nombres binaires, chaque bit remplit la fonction suivante:



Le chevauchement de lutins sera expliqué au Chapitre 8.

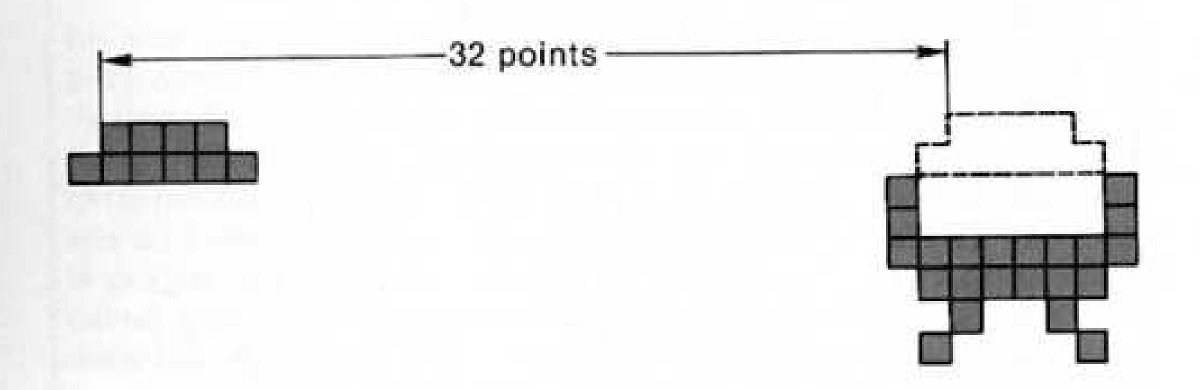
Quand B7 est réglé à 1 et que le code couleur (B3—B0) est réglé à 0010 (soit 2, le vert médium), la donnée devient 10000010. Comme 10000010 correspond à &H82 en hexadécimal, quand

```
COLOR SPRITE$(0)=CHR$(&H82)
```

est exécuté, la première ligne du lutin sur le plan sprite 0 devient le vert médium et la ligne est déplacée de 32 points vers la gauche. Quand le programme suivant est exécuté, le hublot de l'OVNI s'ouvre et un "martien" apparaît.

```
10 SCREEN 5,1
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 SPRITE$(1)=CHR$(&H58)+CHR$(&H58)+CHR$
(&H7E)+CHR$(&H1A)+CHR$(&H18)+CHR$(&H18)+
CHR$(&H0)+CHR$(&H0)
40 CLS
50 PUT SPRITE 0,(120,100),1,0
60 FOR T=0 TO 1000:NEXT T
70 COLOR SPRITE$(0)=CHR$(&H81)+CHR$(&H81
)
80 FOR T=0 TO 1000:NEXT T
90 PUT SPRITE 1,(120,96),14,1
100 GOTO 100
```

La ligne 70 déplace de 32 points vers la gauche la première et la deuxième ligne du lutin (N° de motif 0), affiché sur le plan sprite 0.

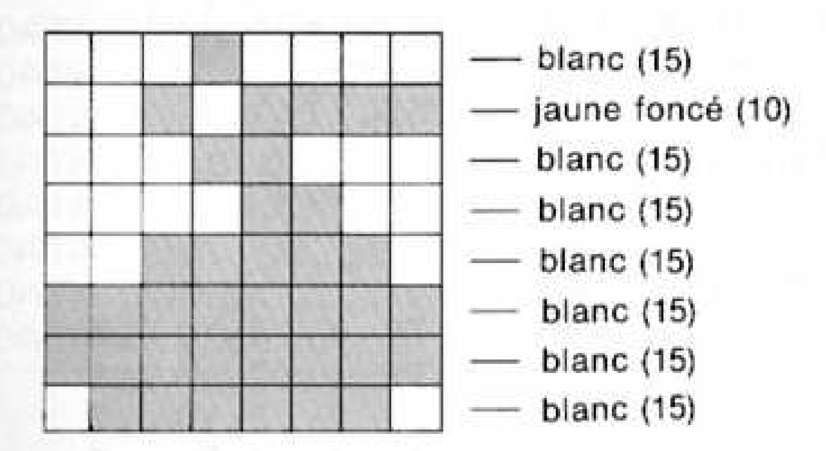


TECHNIQUE DE DEFINITION D'UN SPRITE

La méthode fondamentale de définition d'un motif sprite est d'utiliser la fonction CHR$ pour assigner la donnée de motif à la variable SPRITE$. Dans le programme suivant, la donnée de motif sprite est écrite dans les énoncés DATA et affectée à la variable SPRITE$ par l'énoncé READ. La donnée de motif sprite est écrite dans les énoncés DATA en faisant appel aux points (.) et à la lettre majuscule O afin de fournir une visualisation de la forme du lutin.

```
10 SCREEN 5,1
20 SP$="":SC$=""
30 FOR SI=0 TO 7
40 READ SQ$,SC:SP=0
50 FOR SJ=1 TO 8
60 SP=SP*2-(MID$(SQ$,SJ,1)="O")
70 NEXT SJ
80 SP$=SP$+CHR$(SP):SC$=SC$+CHR$(SC)
90 NEXT SI
100 SPRITE$(0)=SP$
110 COLOR SPRITE$(0)=SC$
120 PUT SPRITE 0,(120,90),,0
130 GOTO 130
140 '
150 DATA ...O....,15
160 DATA ..O.OOOO,10
170 DATA ...OO...,15
180 DATA ....OO..,15
190 DATA ..OOOOO.,15
200 DATA OOOOOOOO,15
210 DATA OOOOOOOO,15
220 DATA .OOOOOO.,15
```

Dans les énoncés DATA des lignes 150 à 220, le lutin à définir est défini par des points et des O. Un point indique un carré non marqué, tandis qu'un O représente un carré marqué. Le numéro, suivant chaque énoncé DATA, est le code couleur qui spécifie la couleur de cette ligne du lutin. Les lignes de 20 à 110 affectent les données dans les énoncés DATA à la variable SPRITE$ et elles spécifient les couleurs par l'énoncé COLOR SPRITE$. A la ligne 60, une fonction (expliquée au Chapitre 7) est utilisée pour changer la donnée en valeur hexadécimale. Ensuite, les lignes 100 et 110 définissent le motif et les couleurs. Le lutin suivant est ainsi défini.



En analysant la méthode utilisée pour définir le motif sprite dans ce programme, on aura l'impression qu'il est fait appel à une technique de programmation avancée, mais en réalité elle est très simple si on la considère comme une formule de définition du motif. L'emploi de cette méthode allonge quelque peu le programme, mais en revanche, elle a l'avantage de fournir une représentation visuelle du lutin dans le programme, pratique lors d'une vérification; de plus, elle évite de devoir convertir chaque donnée en valeur hexadécimale pendant la rédaction du programme. Enfin, toute modification éventuelle du programme est facile car il suffit de changer un point en un O ou un O en un point.

Exemple de programme

Le programme suivant utilise la technique de l'énoncé DATA pour définir les lutins. Ce programme emploie deux énoncés (DEFINT et DEFFN) qui ne seront pas expliqués dans ce manuel. Pour une explication de ces énoncés, prière de consulter le manuel de référence de programmation.

Le programme suivant affiche une cane et ses canetons, nageant sur un étang.

```
10 SCREEN 5,1
20 DEFINT A-Z:X=0:Y=0:Z=0
30 DEFFNX=(Z+240)MOD 256
40 ' *** sprite definition ***
50 RESTORE 380:SN=0:GOSUB 260
60 RESTORE 470:SN=1:GOSUB 260
70 RESTORE 470:SN=2:GOSUB 260
80 RESTORE 470:SN=3:GOSUB 260
90 RESTORE 470:SN=4:GOSUB 260
100 RESTORE 470:SN=5:GOSUB 260
110 ' *** draw pond ***
120 LINE (0,116)-(255,116),7
130 PAINT (0,118),7
140 ' *** move sprite ***
150 Y=100
160 FOR X=0 TO 255
170  Z=X  :PUT SPRITE 0,(Z,Y),,0
180  Z=FNX:PUT SPRITE 1,(Z,Y),,1
190  Z=FNX:PUT SPRITE 2,(Z,Y),,2
200  Z=FNX:PUT SPRITE 3,(Z,Y),,3
210  Z=FNX:PUT SPRITE 4,(Z,Y),,4
220  Z=FNX:PUT SPRITE 5,(Z,Y),,5
230 NEXT X
240 GOTO 160
250 ' *** sprite definition subroutine *
**
260 SP$="":SC$=""
270 FOR SI=0 TO 7
280 READ SQ$,SC:SP=0
290  FOR SJ=1 TO 8
300   SP=SP*2-(MID$(SQ$,SJ,1)="O")
310  NEXT SJ
320  SP$=SP$+CHR$(SP):SC$=SC$+CHR$(SC)
330 NEXT SI
340 SPRITE$(SN)=SP$
350 COLOR SPRITE$(SN)=SC$
360 RETURN
```

```
370 ' *** sprite data ***
380 DATA ...0....,15
390 DATA ..0.0000,10
400 DATA ...00...,15
410 DATA ....00..,15
420 DATA ..00000.,15
430 DATA 00000000,15
440 DATA 00000000,15
450 DATA .000000.,15
460 ' *** sprite data ***
470 DATA ........,0
480 DATA ........,0
490 DATA ....0...,10
500 DATA ...0.000,8
510 DATA ....00..,10
520 DATA .00..00.,10
530 DATA .000000.,10
540 DATA ..0000..,10
```

Chapitre 7
Utilisation des fonctions

FONCTIONS DE TYPE NUMERIQUE

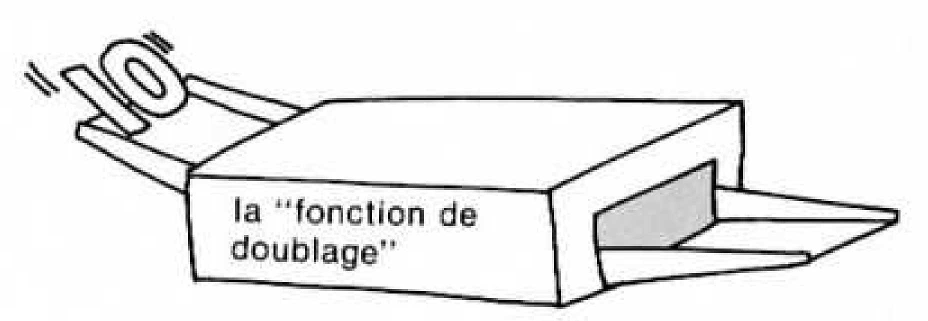
- En quoi consistent les fonctions?
- Diverses fonctions de type numérique
- La fonction aléatoire

EN QUOI CONSISTENT LES FONCTIONS?

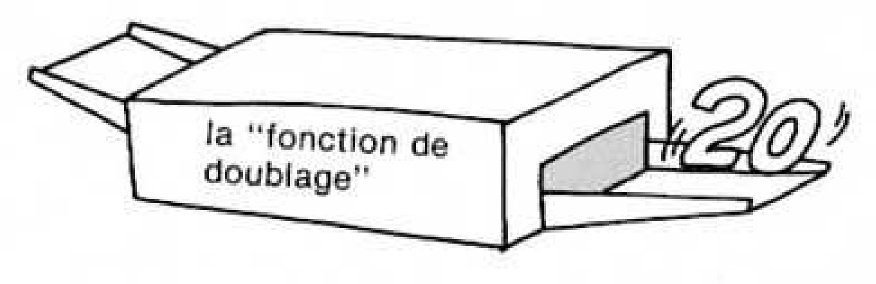
Le langage BASIC comprend à la fois des instructions et des fonctions. Jusqu'à présent, nous nous sommes concentrés sur l'explication de l'emploi des instructions. Désormais, notre attention se portera surtout vers les fonctions.

Une fonction du BASIC peut être considérée comme une boîte qui traite une valeur et donne un résultat.

A titre d'exemple, imaginons que nous disposions d'une fonction qui double un nombre. (En fait, une fonction de ce genre n'existe pas en MSX2-BASIC.) Si le nombre 10 était placé dans cette fonction, il deviendrait 20; dans le cas de 100, on obtiendrait 200, tandis que 450 deviendrait 900, et ainsi de suite. Tout nombre placé dans cette fonction serait donc doublé.



Si 10 est entré,



il en ressort 20.

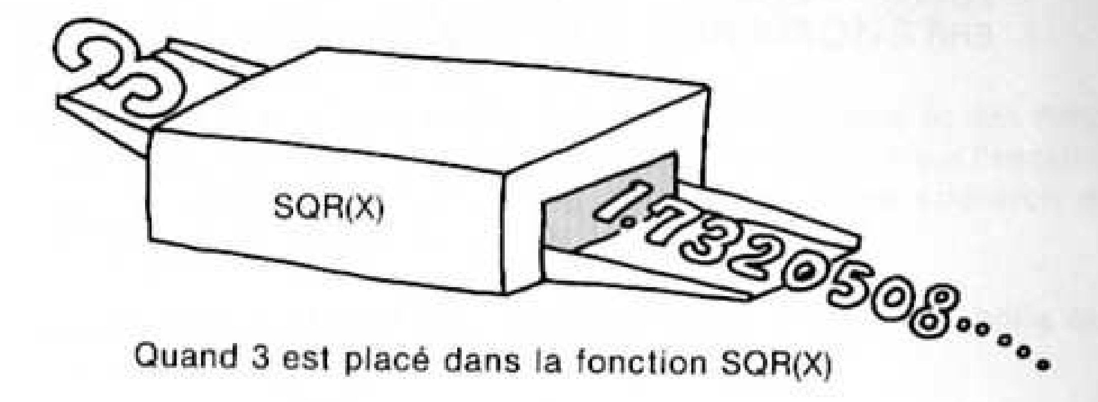
LES FONCTIONS DE TYPE NUMERIQUE

En MSX2-BASIC, les fonctions pour lesquelles l'entrée est un nombre et la sortie est un nombre également sont appelées des fonctions de type numérique. Il existe un total de 17 fonctions de type numérique en MSX2-BASIC.

ABS(X)	CDBL(X)	CINT(X)	CSNG(X)
FIX(X)	INT(X)	SQR(X)	SGN(X)
ATN(X)	COS(X)	SIN(X)	TAN(X)
LOG(X)	EXP(X)	RND(X)	
ERR	ERL		

LA FONCTION RACINE CARREE SQR(X)

Utilisons la fonction SQR(X) afin de mieux comprendre comment agissent les fonctions. Cette fonction-ci extrait la racine carrée du nombre X. Dans le cas des fonctions, au lieu du verbe "extraire", il est plus normal de parler de "rendre". Ainsi, il conviendrait de reformuler cette expression et de dire que la fonction SQR(X) rend la racine carrée de X.
Par exemple, si X est 3, la racine carrée sera 1,7320508 .... Si X est 5, elle sera 2,2360679 ....



Quand 3 est placé dans la fonction SQR(X)

Comment utiliser les fonctions?
Quand 3 est placé dans la fonction SQR(X), la valeur 1,7320508 ... est "rendue". Quand une valeur est entrée dans une fonction, elle y est entrée à la place de X.

```
SQR(3)
```

A présent, 3 a été introduit dans la fonction SQR(X). Mais on pourra également affecter 3 à une variable et entrer la variable dans la fonction à la place du nombre.
Si vous exécutez d'abord

```
A=3
```

avec

```
SQR(A)
```

le nombre 3 sera entré dans la fonction SQR(X).
Dans un cas comme dans l'autre, qu'un nombre soit directement entré, ou qu'une variable soit utilisée, la fonction SQR(X) a maintenant la valeur 1,7320508 .... Chaque fois que l'on appellera cette fonction dans un programme, elle rendra la même valeur 1,7320508 ....

La procédure pour obtenir la valeur d'une fonction dans un programme est la suivante. Comme les fonctions en elles-mêmes ne sont pas des instructions, elles ne peuvent pas servir pour transmettre des ordres à l'ordinateur. Autrement dit, elles doivent être combinées à une instruction pour que la valeur soit "rendue". A cet effet, on fera appel à l'énoncé LET et à l'énoncé PRINT.

```
R=SQR(3)
```

Cet énoncé LET affecte la valeur de SQR(3), à savoir 1,7320508 ... à la variable numérique R. Confirmons que ce résultat est bien obtenu.

```
R=SQR(3)
Ok
PRINT R
 1.7320508075688
Ok
```

La valeur de la racine carrée de 3, jusqu'à 13 décimales soit un total de 14 chiffres, a été affectée à la variable R.
En MSX2-BASIC, tous les résultats de calculs ont normalement une précision de 14 chiffres, mais on pourra aussi utiliser une précision limitée à 6 chiffres. La précision à 14 chiffres est appelée double précision, tandis que celle à 6 chiffres porte le nom de simple précision.

La valeur d'une fonction peut être directement affichée par un énoncé PRINT.

```
PRINT SQR(5)
 2.2360679774998
Ok
```

Les fonctions peuvent également s'employer dans une expression conditionnelle IF—THEN.

```
IF SQR(A)>=10 THEN END
```

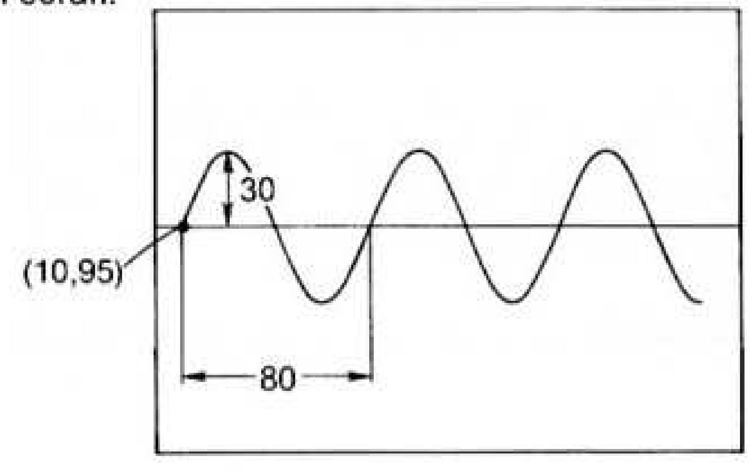
La valeur de SQR(A) change d'après la valeur de A. Dans cet énoncé IF—THEN, si la valeur de SQR(A) devient 10 ou supérieure, le programme est arrêté.
Comme illustré ci-dessus, une fonction traite une valeur entrée selon des règles fixes, d'après ce que la fonction est supposée faire, et elle "rend" le résultat. La valeur rendue est utilisée en combinaison avec certaines instructions du BASIC, telles que l'énoncé LET, l'énoncé PRINT et l'énoncé IF—THEN.

FONCTIONS TRIGONOMETRIQUES SIN(X)

La fonction trigonométrique SIN(X) est une des fonctions numériques. Cette fonction "rend" le sinus de X. Les autres fonctions trigonométriques du MSX2-BASIC sont le cosinus COS(X), la tangente TAN(X) et l'arc tangent ATN(X).
Utilisons la fonction SIN(X) dans un programme qui tracera une courbe sinusoïdale.

```
10 SCREEN 2:CLS
20 PI=3.14
30 LINE (0,95)-(252,95)
40 FOR X=10 TO 230
50 R=(X-10)*PI/40
60 S=SIN(R)
70 VY=S*30
80 Y=95-VY
90 PSET (X,Y)
100 NEXT X
110 GOTO 110
```

Lorsque des fonctions trigonométriques, telles que SIN(X), COS(X) et TAN(X) sont utilisées, la valeur de X est exprimée en radians (1 radian = π). Dans ce programme, X a des valeurs allant de 10 à 230. Ces valeurs sont converties en radians à la ligne 50. Quand la valeur de X est 90, elle est de 2 radians (2π); quand la valeur est 50, on arrive à 1 radian et quand elle atteint 10, le résultat est 0 radian. (X représente la coordonnée X sur l'écran quand la courbe sinusoïdale est tracée.)
Les sinus de ces valeurs en radians sont affectés à S par la fonction SIN(X) à la ligne 60. Quand la valeur du sinus est 1, la distance sur l'axe des Y est 30 (ligne 70). A la ligne 80, la coordonnée Y est déterminée en prenant 95 comme point de référence de la coordonnée Y.
Quand ce programme est exécuté, la courbe sinusoïdale suivante est tracée sur l'écran.



LA FONCTION DE VALEUR ABSOLUE ABS(X)

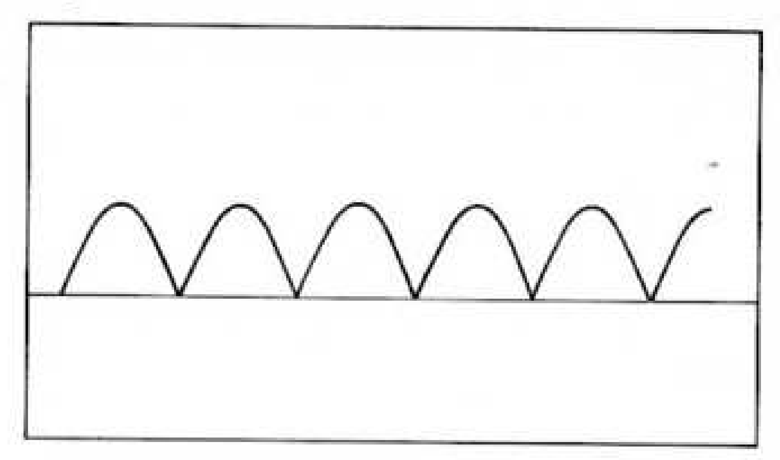
La fonction ABS(X) "rend" la valeur absolue de X.
Par exemple, quand X est 100, la fonction ABS(X) rend la valeur de 100. Quand X est −100, elle rend également la valeur de 100.

```
PRINT ABS(-3.5)
 3.5
Ok.
PRINT ABS(-10)+ABS(8)
 18
Ok
```

Utilisons à présent la fonction ABS(X) dans le programme précédent de la courbe sinusoïdale.

```
10 SCREEN 2:CLS
20 PI=3.14
30 LINE (0,95)-(252,95)
40 FOR X=10 TO 230
50 R=(X-10)*PI/40
60 S=SIN(R)
70 VY=ABS(S*30)
80 Y=95-VY
90 PSET (X,Y)
100 NEXT X
110 GOTO 110
```

Le programme a été modifié de telle sorte que la valeur absolue de SIN(R) * 30 soit affectée à VY à la ligne 70. En conséquence, on obtient la figure suivante sur l'écran.



LA FONCTION ALEATOIRE RND(X)

La fonction RND(X) **rend un nombre aléatoire de 0 à moins de 1** quand X est un nombre plus grand que 0.
Exécutez

```
PRINT RND(1)
```

plusieurs fois en mode direct.

```
PRINT RND(1)
 .59521943994623
Ok
PRINT RND(1)
 .10658628050158
Ok
PRINT RND(1)
 .76597651772823
```

Chaque fois que la fonction RND(X) est exécutée, elle rend un nombre aléatoire de 0 à moins que 1. La fonction RND(X) peut servir à produire des nombres aléatoires dans une plage déterminée.

LA FONCTION RND(X) ET LA FONCTION NOMBRE ENTIER INT(X)

Etudions à présent comment utiliser la fonction RND(X) pour rendre des nombres entiers aléatoires de 1 à 200. Par elle-même, la fonction RND(X) ne fournira que des valeurs comprises entre 0 et moins que 1 (de 0 à 0,99999999999999). Si nous multiplions RND(1) par 200

```
RND(1)*200
```

la valeur sera un nombre compris entre 0 et moins que 200. Mais cette valeur sera un nombre de 14 chiffres, y compris ceux qui sont placés après la virgule. La fonction de type numérique INT(X) est destinée à changer cette valeur en un nombre entier.
La fonction INT(X) convertit toute valeur de X en un nombre entier et elle "rend" le nombre entier.

```
PRINT INT (12.73)
 12
Ok
PRINT INT (-7.99)
-8
Ok
```

La fonction INT(X) rend la valeur entière maximale, plus petite que la valeur de X. En conséquence, si RND(1) * 200 est utilisé comme valeur de INT(X),

```
INT(RND(1)*200)
```

des nombres aléatoires de 0 à moins que 200 (de 0 à 199) seront rendus. Dès lors, pour rendre des nombres aléatoires de 0 à 200, il suffira d'utiliser

```
INT((RND(1)*(200+1))
```

ou

```
INT (RND(1)*201)
```

```
10 FOR L=1 TO 10
20 X=INT(RND(1)*201)
30 PRINT X;
40 NEXT L
```

Ce programme produira et affichera 10 nombres aléatoires de 0 à 200, comme illustré ci-dessous.

```
RUN
 119  21  153  116  147  37  74  190
 128  94
```

Le format standard pour produire un nombre aléatoire de 0 à N est le suivant:

X = INT(RND(1) * (N + 1))

Par ce format, un nombre aléatoire de 0 à N sera affecté à X.

Le format suivant s'emploie pour produire un nombre aléatoire de M à N.

X = INT(RND(1) * (N – M + 1)) + M

Par exemple, pour produire un nombre aléatoire de 2 à 15, M devrait être 2 et N devrait être 15. Par conséquent, N – M + 1 = 14.

X = INT(RND(1) * 14) + 2

Les boîtes aléatoires

Rédigeons un programme qui tracera 50 carrés de tailles et de couleurs différentes en des endroits différents de l'écran.

```
10 SCREEN 5
20 COLOR ,1,1:CLS
30 FOR B=1 TO 50
40 SX=INT(RND(1)*220)+5
50 SY=INT(RND(1)*180)+5
60 ST=INT(RND(1)*40)+20
70 C=INT(RND(1)*14)+2
80 LINE (SX,SY)-STEP(ST,ST),C,BF
90 NEXT B
100 GOTO 100
```

Les variables SX,SY sont les coordonnées du coin supérieur gauche de chaque carré, ST représente la longueur d'un des côtés et C détermine le code couleur. Les lignes de 40 à 70 affectent des nombres aléatoires à ces variables par la fonction RND(X). La plage des valeurs pour chacun des nombres aléatoires est:

SX...de 5 à 224
SY...de 5 à 184
ST...de 20 à 59
C...de 2 à 15

La ligne 80 trace les carrés de tailles et de couleurs différentes à des endroits d'affichage différents.

Production de nombres aléatoires différents à chaque exécution du programme

Comme on peut le voir, si le programme des "boîtes aléatoires" ci-dessus est exécuté plusieurs fois, les carrés seront toujours disposés aux mêmes endroits. En fait, l'ordinateur comporte une liste de nombres aléatoires et, à chaque exécution d'un programme utilisant la fonction RND(1), cette fonction rend les valeurs de cette liste dans le même ordre, en commençant par la première de la liste en question.

Toutefois, il est possible de rédiger un programme qui rend, à chaque exécution, une partie différente de la liste des nombres aléatoires; de ce fait, on pourra, bien entendu, obtenir des résultats différents chaque fois que le programme sera exécuté.

Une manière d'y arriver consiste à faire appel au rythmeur interne de l'ordinateur en ajoutant les quelques lignes suivantes au début du programme.

```
22 FOR N=0 TO TIME-INT(TIME/100)*100
24 X=RND(1)
26 NEXT N
```

La variable TIME de la ligne 22 est une variable spéciale du MSX-BASIC à laquelle la valeur actuelle du rythmeur interne est toujours affectée. La valeur du rythmeur interne change de 0 à 65535, par unités de 1, environ tous les 1/50 de seconde. Lorsque la valeur atteint 65535, elle repasse à 0 et le processus se répète.
Lorsqu'un programme, contenant des lignes telles que celles qui sont indiquées ci-dessus, est exécuté, la valeur de la variable TIME sera un nombre quelconque compris entre 0 et 65535. Cependant, la ligne 22 contient deux variables TIME et, comme la valeur du rythmeur interne change à une vitesse relativement élevée, les valeurs affectées à chacune de ces deux variables TIME seront légèrement différentes. Par exemple, si une valeur de 42280 est affectée à la première variable TIME, une valeur d'environ 42281 le sera à la seconde variable TIME. En conséquence, la ligne 22 devrait être

```
FOR N=0 TO 42280-INT(42281/100) * 100
```

ou

```
FOR N=0 TO 80
```

Cependant, si la valeur affectée à la première variable TIME est 10900, et que celle qui est affectée à la seconde variable TIME est 10901, l'énoncé devrait devenir:

```
FOR N=0 TO 10900-INT(10901/100) * 100
```

ou

```
FOR N=0 TO 0
```

De cette façon, la valeur finale de N dans l'énoncé FOR—NEXT quand le programme est exécuté changera selon la valeur du rythmeur interne. Quand la valeur finale de N est 80, X=RND(1) à la ligne 24 sera exécuté 81 fois par la boucle FOR—NEXT. En conséquence, la fois suivante où la fonction RND(1) sera utilisée dans le programme, elle fournira la 82ème valeur de la liste des nombres aléatoires dont dispose l'ordinateur.
Ajoutons donc ces trois lignes au programme précédent des "boîtes aléatoires".

```
10 SCREEN 5
20 COLOR ,1,1:CLS
22 FOR N=0 TO TIME-INT(TIME/100)*100
24 X=RND(1)
26 NEXT N
30 FOR B=1 TO 50
40 SX=INT(RND(1)*220)+5
50 SY=INT(RND(1)*180)+5
60 ST=INT(RND(1)*40)+20
70 C=INT(RND(1)*14)+2
80 LINE (SX,SY)-STEP(ST,ST),C,BF
90 NEXT B
100 GOTO 100
```

Mais il existe une autre méthode pour obtenir des nombres aléatoires différents à chaque exécution d'un programme. Cette méthode fait appel à la fonction INKEY$ qui permet l'entrée de données par le clavier. Pour utiliser la fonction INKEY$, on devra ajouter les trois lignes suivantes au programme.

```
22 A$=INKEY$
24 X=RND(1)
26 IF A$="" THEN 22
```

La fonction INKEY$ sera expliquée en détails à la page 206. Qu'il suffise ici de savoir que, grâce à ces trois lignes, X = RND(1) continuera d'être exécuté jusqu'à ce que soit actionnée une touche du clavier. Ajoutez ces trois lignes au programme des "boîtes aléatoires" et vérifier-en les résultats.

FONCTIONS DE TYPE EN CHAINE

- En quoi consistent les fonctions de type en chaîne?
- Traitement des chaînes de caractères

EN QUOI CONSISTENT LES FONCTIONS DE TYPE EN CHAINE?

Une fonction de type en chaîne traite une chaîne de caractères et elle rend le résultat sous forme d'une chaîne de caractères.

En MSX2-BASIC, il existe sept fonctions de type en chaîne.
LEFT$(X$,N), MID$(X$,M[,N]), RIGHT$(X$,N)
SPACE$(N), STRING$(N,J) ou STRING$(N,X$),
TAB(N), SPC(N)

Plusieurs de ces fonctions acceptent une entrée à valeur numérique et elles "rendent" une chaîne de caractères, mais elles sont classées comme fonctions de type en chaîne.

SPECIFICATION DES ESPACES SPACE$(N), SPC(N)

Les fonctions SPACE$(N) et SPC(N) fournissent un nombre N d'espaces vierges de caractères. Ces deux fonctions apportent les mêmes résultats, mais SPC(N) ne peut être utilisée qu'avec un énoncé PRINT.

Voyons comment ces fonctions agissent en mode direct.

```
PRINT "A";SPACE$(10);"B"
A          B          10 espaces
Ok
PRINT "C";SPC(15);"D"
C               D     15 espaces
Ok
S$=SPACE$(5)
Ok
PRINT "X";S$;"Y";S$;"Z"
X     Y     Z         5 espaces
Ok
```

Une fonction de type en chaîne fournit toujours des caractères. Par conséquent, quand la valeur est affectée avec un énoncé LET, la variable doit également être une variable de type en chaîne.
Dans l'exemple précédent,

```
S$=SPACE$(5)
```

affecte cinq espaces à la variable en chaîne S$.

TRAITEMENT DES CHAINES DE CARACTERES
LEFT$(X$,N), MID$(X$, M, N), RIGHT$(X$, N)

Les fonctions de type en chaîne LEFT$(X$, N), MID$(X$, M, N) et RIGHT$(X$, N) rendent une partie d'une chaîne de caractères spécifiée.

LEFT$(X$,N) fournit le nombre N de caractères à compter de la gauche de la chaîne de caractères X$.

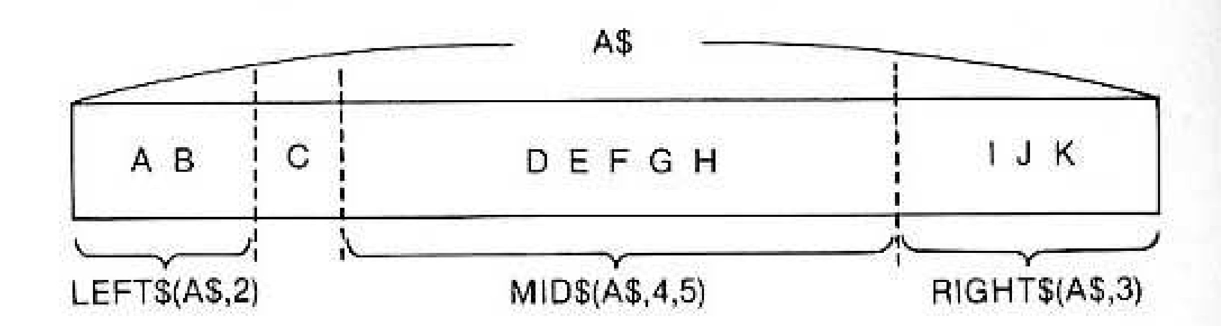
RIGHT$(X$,N) fournit le nombre N de caractères à compter de la droite de la chaîne de caractères X$.

MID$(X$,M,N) fournit le nombre N de caractères à compter du M-ième caractère de la chaîne de caractères.

Les énoncés PRINT suivants en mode direct illustrent l'opération de ces trois fonctions.

```
A$="ABCDEFGHIJK"
Ok
PRINT LEFT$(A$,2)
AB
Ok
PRINT RIGHT$(A$,3)
IJK
Ok
PRINT MID$(A$,4,5)
DEFGH
Ok
```

Comme on le voit, les fonctions LEFT$(X$,N), MID$(X$, M,N) et RIGHT$(X$,N) permettent d'obtenir une partie spécifiée d'une chaîne de caractères.



Dans le programme suivant, toutes les chaînes de caractères (noms de personnes), entrées par l'énoncé INPUT, sont affichées; ensuite, la fonction LEFT$(X$,N) est employée pour afficher séparément tous les noms commençant par la lettre J, c'est-à-dire toutes les chaînes de caractères où J est le premier caractère sur la gauche.

```
10 DIM N$(10):Y=1
20 CLS
30 FOR L=1 TO 10
40 INPUT "Name";N$(L)
50 NEXT L
60 CLS
70 FOR L=1 TO 10
80 LOCATE 2,L:PRINT N$(L)
90 NEXT L
100 FOR L=1 TO 10
110 A$=LEFT$(N$(L),1)
120 IF A$<>"J" THEN 150
130 LOCATE 15,Y:PRINT N$(L)
140 Y=Y+1
150 NEXT L
160 LOCATE 0,20
```

L'énoncé INPUT de la ligne 40 affecte les noms ou prénoms à la variable en tableau N$(1)—N$(10). Par exemple, les prénoms suivants peuvent être affectés dans l'ordre en commençant par N$(1):
PETER, PAUL, JACK, MARY, SUSIE, JOHN, JANE, TOM, DICK, CATHARINE

Après affichage de tous les prénoms par les lignes 70 à 90, les lignes 100 à 150 affichent uniquement ceux qui commencent par J, à savoir JACK, JOHN et JANE, sur le côté droit de l'écran.

```
PETER          JACK
PAUL           JOHN
JACK           JANE
MARY
SUSIE
JOHN
JANE
TOM
DICK
CATHARINE
```

La fonction LEFT$(X$,N) est employée à la ligne 110 pour afficher uniquement les prénoms, commençant par J. Ici, seul le premier caractère sur la gauche de la chaîne de caractères, affectée à la variable en tableau N$, est affecté en A$. Ensuite, l'énoncé IF—THEN de la ligne 120 vérifie si ce caractère est "J". S'il s'agit bien de J, le contenu de N$ est affiché sur le côté droit de l'écran.

FONCTIONS DE CONVERSION DES DONNEES NUMERIQUES ET DE TYPE EN CHAINE

- Les fonctions de conversion
- Le code de caractères

LES FONCTIONS DE CONVERSION

Les fonctions de conversion sont classées selon qu'elles fournissent une donnée de type en chaîne quand une donnée numérique est entrée ou qu'elles fournissent une donnée numérique quand une donnée de type en chaîne est entrée. En MSX2-BASIC, il existe neuf fonctions de conversion, à savoir:

ASC(X$), CHR$(X)
VAL(X$), STR$(X)
LEN(X$), INSTR([N,]X$, Y$)
BINS(X), OCT$(X), HEX$(X)

CHANGEMENT DU TYPE DE NOMBRES
VAL(X$), STR$(X)

En BASIC, tout nombre peut être traité soit comme valeur numérique, soit comme nombre de type en chaîne. Ainsi, par exemple, si 123 est affecté à une variable de type numérique, comme dans le cas suivant,

```
A=123
```

alors, 123 sera considéré comme le nombre cent-vingt-trois. Mais, si 123 est affecté à une variable en chaîne, comme dans le cas ci-dessous,

```
A$="123"
```

alors, 123 sera considéré comme les caractères un, deux et trois.

VAL(X$) change un nombre traité comme caractères en une valeur de type numérique.
STR$(X) change une nombre traité comme valeur numérique en un nombre de type en chaîne.

Entrons le programme suivant:

```
10 A$="123":B$="456"
20 X=VAL(A$):Y=VAL(B$)
30 PRINT "A$+B$=";A$+B$
40 PRINT "VAL(A$)+VAL(B$)=";X+Y
```

Tout d'abord, les nombres 123 et 456 sont affectés aux variables A$ et B$ comme valeurs de type en chaîne. Ensuite, à la ligne 20, ils sont convertis en valeurs numériques. Les lignes 30 et 40 affichent la somme des nombres de type en chaîne et des valeurs numériques. L'illustration suivante indique l'affichage obtenu à l'exécution de ce programme.

```
RUN
A$+B$=123456
VAL(A$)+VAL(B$)= 579
OK
```

La fonction STR$(X) remplit le rôle inverse de VAL(A$). Lançons le programme suivant:

```
10 A=123:B=456
20 X$=STR$(A):Y$=STR$(B)
30 PRINT "A+B=";A+B
40 PRINT "STR$(A)+STR$(B)=";X$+Y$
RUN
A+B= 579
STR$(A)+STR$(B)= 123 456
OK
```

Quand une valeur numérique a été changée en un nombre de type en chaîne, l'espace précédant la valeur numérique (où apparaît le signe + ou −) est inclus comme une partie de la valeur de type en chaîne.

CODES DE CARACTERES ET FONCTIONS ASC(X$), CHR$(X)

Tout comme il existe des codes pour les couleurs, le BASIC a prévu des codes de caractères. A titre d'exemple, disons que le code caractère pour le A majuscule est 65 (nombre décimal). Il existe deux fonctions, en MSX2-BASIC, qui font appel aux codes de caractères.

ASC(X$) fournit le code de caractère d'un caractère entré.
CHR$(X) fournit le caractère d'un code de caractère entré.

Ces fonctions peuvent être vérifiées comme suit en mode direct:

```
PRINT ASC("A")     } affiche le code
 65                } caractère de A
Ok
PRINT CHR$(66)     } affiche le caractère (B)
B                  } correspondant au code
Ok                 } caractère 66
```

On trouvera la liste des codes de caractères dans le manuel de référence de programmation en MSX2-BASIC.

OBTENTION DE LA LONGUEUR D'UNE CHAINE DE CARACTERES LEN(X$)

La fonction LEN(X$) fournit le nombre de caractères d'une chaîne sous forme de valeur numérique.

```
A$="ABCDE"
Ok
PRINT LEN(A$)
 5
Ok
```

Le nombre de caractères, à savoir 5, de la chaîne de caractères "ABCDE" est fourni par LEN(A$).

Le programme suivant utilise les fonctions LEN(X$), MID$(X$,M,N) et CHR$(X) pour convertir chaque caractère, affecté à la variable A$ par l'énoncé INPUT, en un caractère dont le numéro de code est d'une unité supérieur à celui qui a été entré.

```
10 CLS
20 INPUT "Any letters";A$
30 N=LEN(A$)
40 FOR L=1 TO N
50 B$=MID$(A$,L,1)
60 X=ASC(B$)
70 C$=CHR$(X+1)
80 PRINT C$;
90 NEXT L
100 END
```

Essayez d'entrer des lettres de votre choix et observez les résultats.

```
RUN
Any letters? RNMX
SONY
Ok
RUN
Any letters? LRW
MSX
Ok
```

FONCTIONS D'ENTREE DE DONNEES

- En quoi consistent les fonctions d'entrée de données?
- Entrée de données via le clavier
- Entrée de l'état de la touche de curseur

OPERATION DES FONCTIONS D'ENTREE DE DONNEES

Toutes les fonctions étudiées jusqu'à présent avaient pour mission de traiter une valeur entrée et de donner le résultat. Celles que nous rencontrerons ici sont d'un genre légèrement différent. Dans ces fonctions, les valeurs entrées ne sont pas directement traitées. En effet, les valeurs qui y sont entrées sont comme des signes, portant une signification particulière. Ces signes disent aux fonctions d'entrer l'état des dispositifs d'entrée, raccordés à l'ordinateur, et les fonctions fournissent alors cet état sous forme de donnée. (Certaines de ces fonctions d'entrée de données ne requièrent aucune valeur d'entrée.)

Ces fonctions ont l'avantage de permettre d'écrire des programmes qui exerceront une action en se basant sur l'état d'un dispositif d'entrée; ce sera le cas, par exemple, du déplacement d'un "lutin" dans le sens d'une des touches de curseur qui aura été actionnée.

Les fonctions d'entrée de données

Le MSX2-BASIC possède 22 fonctions d'entrée de données, à savoir:

- Entrée depuis l'écran
  CSRLIN, POS(X), POINT(X,Y)
- Entrée depuis l'imprimante
  LPOS(X)
- Entrée depuis la mémoire
  FRE(X), FRE(" "), PEEK(N), VARPTR(variable), VPEEK(N)
- Entrée depuis le clavier
  INKEY$, INPUT$(X)
- Entrée depuis le port E/S
  INP(N)
- Entrée depuis un levier de commande, barre d'espacement, souris, manette de mouvement, poignée de jeux, clavier tactile ou photostyle
  STICK(N), STRIG(N), PDL(N), PAD(N)
- Entrée depuis un fichier de données
  EOF(numéro fichier), INPUT$(N,[#]numéro fichier)
- Entrée depuis un disque
  DSKF(numéro disque), LOC(numéro fichier),
  VARPTR(#numéro fichier)
- Entrée depuis un sous-programme en langage-machine
  USR[X](I)

ENTREE DE DONNEES VIA LE CLAVIER INKEY$

Quand une touche est actionnée sur le clavier, la fonction INKEY$ fournit le caractère de la touche pressée sous forme d'une donnée de type en chaîne.
Le petit programme suivant illustre l'opération de la fonction INKEY$.

```
10 K$=INKEY$
20 PRINT K$;
30 GOTO 10
```

Quand ce programme est exécuté,

```
K$=INKEY$
```

à la ligne 10, sera exécuté de façon répétée. Si aucune touche n'est actionnée quand la ligne 10 est exécutée, la fonction INKEY$ fournit une chaîne de caractères sans données, ce que l'on appelle une **chaîne vide**. Dans le programme ci-avant, quand une chaîne vide est obtenue, elle est affectée à K$. Cependant, comme il n'existe aucune donnée dans une chaîne vide, quand l'énoncé PRINT de la ligne 20 affiche la chaîne vide, l'effet sur l'écran est le même que si rien ne s'était produit.
Quand une touche comme la touche [A] est actionnée pendant que la ligne 10 est exécutée, la fonction INKEY$ fournit le caractère A. (Si la touche du [a] minuscule a été actionnée, on obtiendra le "a" minuscule.) Ce caractère est affecté à K$, affiché par la ligne 20 et c'est ainsi que le caractère A est affiché sur l'écran.

```
RUN
A
```

Quand [A] est actionné

Quand d'autres touches sont actionnées, les caractères qui y correspondent sont également affichés. L'écran ressemble à une situation d'instruction-attente, mais le curseur n'est pas affiché et ceci indique qu'un programme est en cours d'exécution. En outre, si la touche [↵] est actionnée, le caractère suivant sera affiché au début de la même ligne, au lieu de l'être sur la ligne suivante.
Pour arrêter le programme, utilisez [CTRL] + [STOP].

Préparons un autre programme, utilisant la fonction INKEY$.

```
10 CLS
20 LOCATE 5,4
30 PRINT "┌──────────────────┐"
40 LOCATE 5,14
50 PRINT "└──────────────────┘"
60 FOR Y=5 TO 13 STEP 2
70 FOR X=6 TO 20
80 K$=INKEY$
90 IF K$="" THEN 80
100 LOCATE X,Y:PRINT K$
110 NEXT X
120 NEXT Y
130 LOCATE 0,22:END
```

Ce programme illustre une des utilisations importantes de la fonction INKEY$. Examinons la combinaison de l'énoncé INKEY$ à la ligne 80 et l'énoncé IF—THEN de la ligne 90.

```
80 K$=INKEY$
90 IF K$="" THEN 80
```

Si aucune touche n'est actionnée quand la ligne 80 est exécutée, une chaîne vide est affectée à K$. Si le contenu de K$ est une chaîne vide, alors la ligne 90 ramène le programme à la ligne 80. Par conséquent, le programme continue la répétition de cette boucle aussi longtemps qu'une touche n'est pas actionnée.
L'absence d'espace entre les deux guillemets ("") dans

```
K$=""
```

indique une chaîne vide.

Quand une touche est actionnée, le programme passe à la ligne 100 et le caractère correspondant à la touche actionnée est ainsi affiché.

Cette fonction INKEY$ s'emploie très souvent de cette façon dans des programmes pour faire avancer le programme à la démarche suivante quand une touche est actionnée.

La fonction INKEY$ peut également s'employer pour faire avancer le programme à la démarche suivante, seulement en réponse à une touche spéciale, comme la barre d'espacement. C'est le cas présenté dans le programme ci-après.

```
10 CLS
20 INPUT "Any letters";A$
30 K$=INKEY$
40 IF K$="" OR K$<>" " THEN 30
50 PRINT A$
```

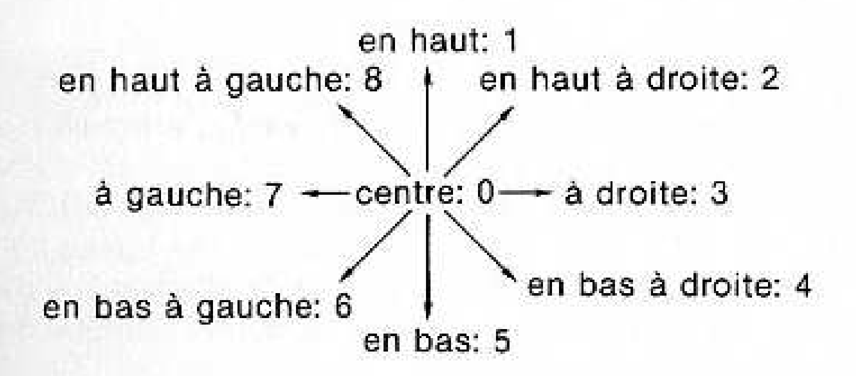
L'énoncé IF—THEN de la ligne 40 ramène le programme à la ligne 30 si: 1) K$ est une chaîne vide, ou si 2) K$ n'est pas la barre d'espacement. Le programme passera à la ligne 50 uniquement si la barre d'espacement est actionnée.

ENTREE DE L'ETAT DE LA TOUCHE DE CURSEUR
STICK(N)

La fonction STICK(N) fournit une valeur numérique qui indique la direction de la touche du curseur, du levier de commande, de la souris, de la poignée de jeux ou du clavier tactile.
La valeur de N détermine si l'état de la touche de curseur ou d'un des autres dispositifs d'entrée est fourni.

N	Dispositif
0	touche de curseur
1	dispositif raccordé à CONTROLLER A
2	dispositif raccordé à CONTROLLER B

La fonction STICK(N) rend les valeurs de 0 à 8, indiquant la direction de la touche de curseur ou des autres dispositifs.



Par exemple, pour STICK(0), 0 sera obtenu quand aucune touche du curseur n'est actionnée.
Quand la touche (en haut) est actionnée, 1 est obtenu; quand (en bas) est actionnée, c'est 5 qui est rendu; et quand et sont actionnées ensemble, la valeur 2 est fournie.

Utilisons à présent la fonction STICK(N) pour réaliser un programme dans lequel les touches du curseur contrôlent l'affichage sur l'écran.

```
10 CLS
20 X=14:Y=10
30 LOCATE X,Y:PRINT "o"
40 C=STICK(0)
50 IF C=0 THEN 40
60 IF C=1 THEN VX=0:VY=-1:GOSUB 110
70 IF C=3 THEN VX=1:VY=0:GOSUB 110
80 IF C=5 THEN VX=0:VY=1:GOSUB 110
90 IF C=7 THEN VX=-1:VY=0:GOSUB 110
100 GOTO 40
110 X=X+VX:Y=Y+VY
120 IF X>29 THEN X=29
130 IF X<0 THEN X=0
140 IF Y>21 THEN Y=21
150 IF Y<0 THEN Y=0
160 LOCATE X,Y:PRINT "o"
170 RETURN
```

Dans ce programme, la lettre "o" minuscule est affichée d'après le sens du mouvement de la touche du curseur. Le "o" est d'abord affiché à l'emplacement 14,10 (lignes 20,30). La fonction STICK(0) de la ligne 40 fournit l'état de la touche du curseur. Les lignes de 50 à 90 déterminent les valeurs des variables VX et VY, d'après la valeur fournie par la fonction STICK(0), de façon à spécifier l'emplacement où le "o" suivant devra être affiché.
Par exemple, quand la touche ⇨ est actionnée, la fonction STICK(0) rend la valeur 3 et, à la ligne 70, VX devient 1 et VY devient 0. Ensuite, le programme saute au sous-programme, commençant à la ligne 110.

Dans le sous-programme, l'emplacement d'affichage du "o" suivant est affecté à X et Y et le "o" est ainsi affiché. Les lignes 120 à 150 du sous-programme ont pour but de limiter la dimension de la zone d'affichage.

Chapitre 8 Interruptions

REALISATION D'INTERRUPTIONS

- Qu'est-ce qu'une interruption?
- Les interruptions du MSX2-BASIC
- Réalisation d'interruptions

QU'EST-CE QU'UNE INTERRUPTION?

Une interruption suspend le déroulement d'un programme quand une condition spécifique se produit pendant son exécution; elle accomplit alors un programme de traitement séparé, appelé programme ou routine de traitement d'interruption.
Une interruption est comparable à un sous-programme, sauf qu'un sous-programme est accompli uniquement quand un énoncé GOSUB est exécuté dans le programme.
Par contre, un programme de traitement d'interruption est exécuté par une condition externe, telle qu'une poussée sur la touche [F1].
Le déroulement normal du programme reprend son cours lorsque l'exécution du programme de traitement d'interruption est achevée, exactement comme dans le cas d'un sous-programme.

LES INTERRUPTIONS DU MSX2-BASIC

Il existe cinq manières d'effectuer une interruption en MSX2-BASIC.

- Par une poussée sur une touche de fonction ([F1]—[F10]).
- Par une poussée sur la barre d'espacement ou par action de la souris, du levier de commande, de la poignée de mouvement ou du clavier tactile.
- Par une poussée sur [CTRL]+[STOP].
- Lors d'un chevauchement de lutin (sprite).
- Après écoulement d'une durée spécifiée (à l'aide du rythmeur interne).

REALISATION D'INTERRUPTIONS

Les cinq énoncés de déclaration d'interruption suivants sont employés pour provoquer une interruption de la routine principale pendant l'exécution d'un programme en MSX2-BASIC.

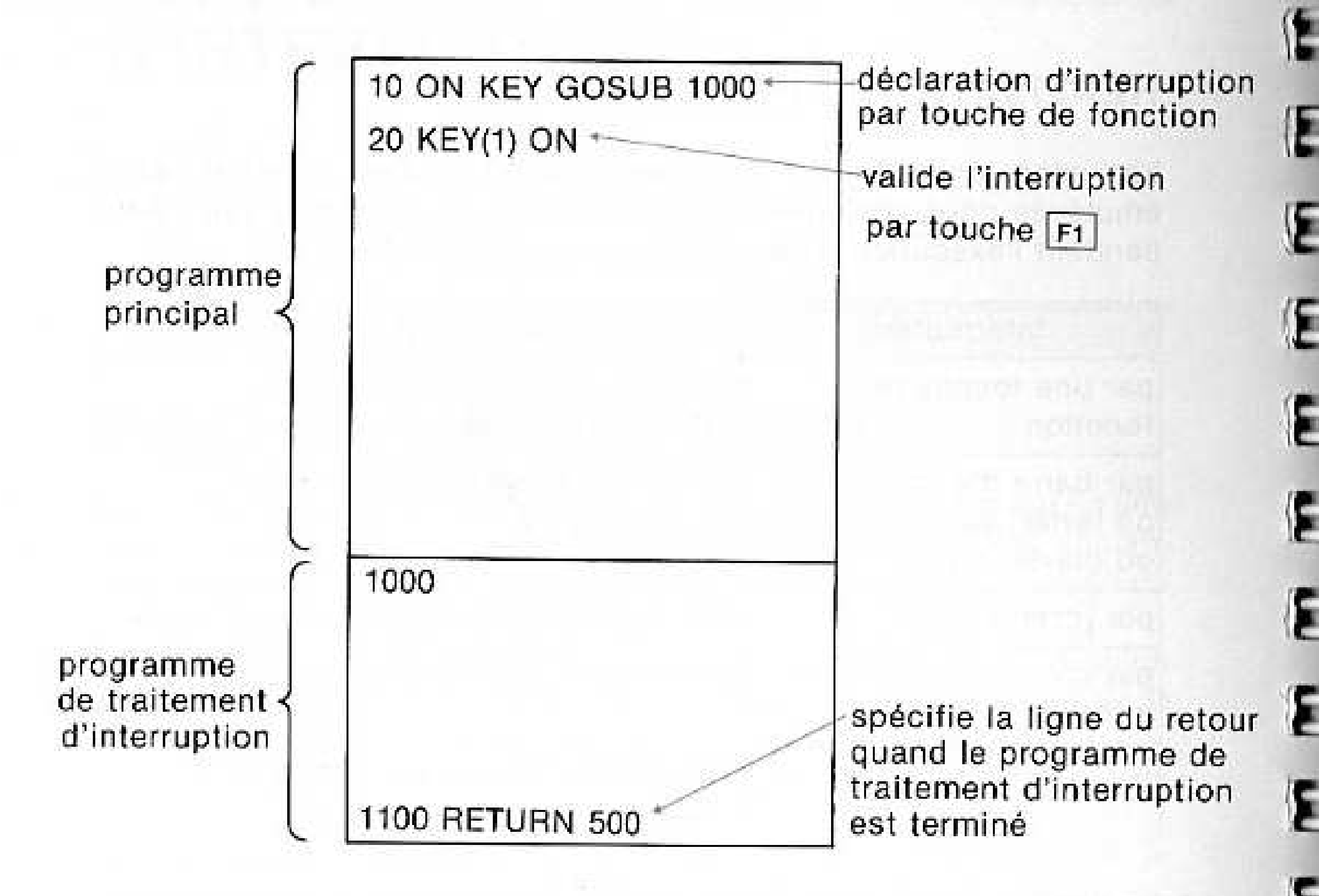
Interruption	Enoncé déclaratif d'interruption
par une touche de fonction	ON KEY GOSUB numéro ligne, [, numéro de ligne]...
par barre d'espacement, ou levier, souris, poignée ou clavier tactile	ON STRIG GOSUB numéro ligne [, numéro ligne]...
par CTRL + STOP	ON STOP GOSUB numéro ligne
par chevauchement de lutin	ON SPRITE GOSUB numéro ligne
par rythmeur interne	ON INTERVAL = durée intervalle GOSUB numéro ligne

Un énoncé déclaratif d'interruption définit ce qui provoquera l'interruption et il spécifie le numéro de ligne de la première ligne du programme de traitement d'interruption.

Un énoncé validant l'interruption est exécuté immédiatement après un énoncé déclaratif d'interruption. Il existe cinq énoncés de validation.

Interruption	Enoncé de validation d'interruption
par une touche de fonction	KEY(N) ON (N est 1—10, 1 = touche F1)
par barre d'espacement, ou levier, souris, poignée ou clavier tactile	STRIG(N) ON (N est 0—4, 0 = barre d'espacement)
par CTRL + STOP	STOP ON
par chevauchement du lutin	SPRITE ON
par rythmeur interne	INTERVAL ON

Le programme suivant exécutera le programme de traitement d'interruption, commençant à la ligne 1000, quand la touche F1 sera actionnée.



Le programme principal est exécuté quand un programme de ce genre est lancé, mais si la touche F1 est actionnée au cours de l'exécution, le programme saute à la ligne 1000 et il exécute le sous-programme de traitement d'interruption. Quand son exécution est terminée, l'énoncé RETURN 500 placé à la fin du programme de traitement fait revenir le programme à la ligne 500 du programme principal.

PROGRAMMES UTILISANT DES INTERRUPTIONS

- Interruptions par touche de fonction
- Invalidation d'une interruption
- Maintien d'une interruption
- Interruption par chevauchement de lutin

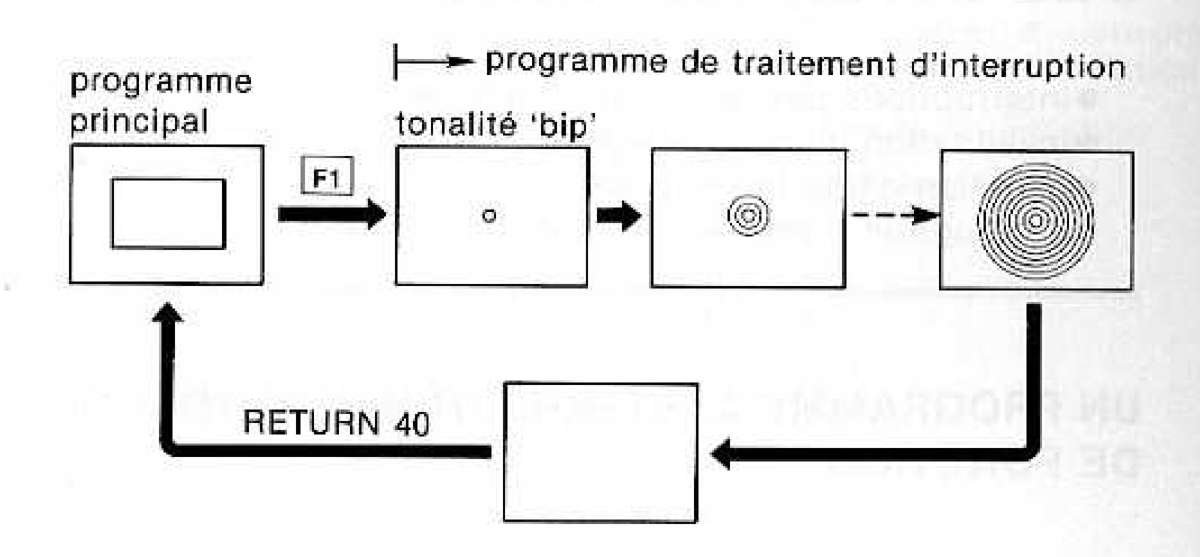
UN PROGRAMME A INTERRUPTION PAR TOUCHE DE FONCTION

Le programme suivant illustre l'emploi de la touche F1 pour provoquer une interruption.

```
10 ON KEY GOSUB 100
20 KEY(1) ON
30 SCREEN 2
40 LINE (50,50)-(200,150),,B
50 GOTO 40
100 'subroutine
110 BEEP:CLS
120 FOR L=10 TO 90 STEP 10
130 CIRCLE (120,100),L
140 NEXT L
150 CLS
160 RETURN 40
```

(lignes 10 à 50 : programme principal ; lignes 100 à 160 : programme de traitement d'interruption)

Les lignes 10 et 20 de ce programme spécifient que le programme passera à la routine de traitement d'interruption commençant à la ligne 100 quand la touche F1 sera actionnée.
Un rectangle est affiché de façon continue par les lignes 40 et 50 du programme principal pendant son exécution. Une interruption se produit quand la touche F1 est actionnée et le programme saute alors à la ligne 100. Une tonalité ('bip') se fait entendre et le rectangle disparaît (BEEP:CLS). Ensuite, 9 cercles concentriques sont tracés, l'écran est vidé après que le dernier cercle est tracé et le programme repasse à la ligne 40.


programme de traitement d'interruption
programme
principal
tonalité 'bip'
F1
RETURN 40

INVALIDATION D'UNE INTERRUPTION
KEY(N) OFF

Ajoutez la ligne suivante au programme précédent.

```
105 KEY(1) OFF
```

Quand est exécuté le programme auquel cette ligne a été ajoutée, une interruption se produit la première fois que la touche F1 est actionnée, mais plus lorsque cette touche est pressée par la suite. La raison en est que

```
KEY(1) OFF
```

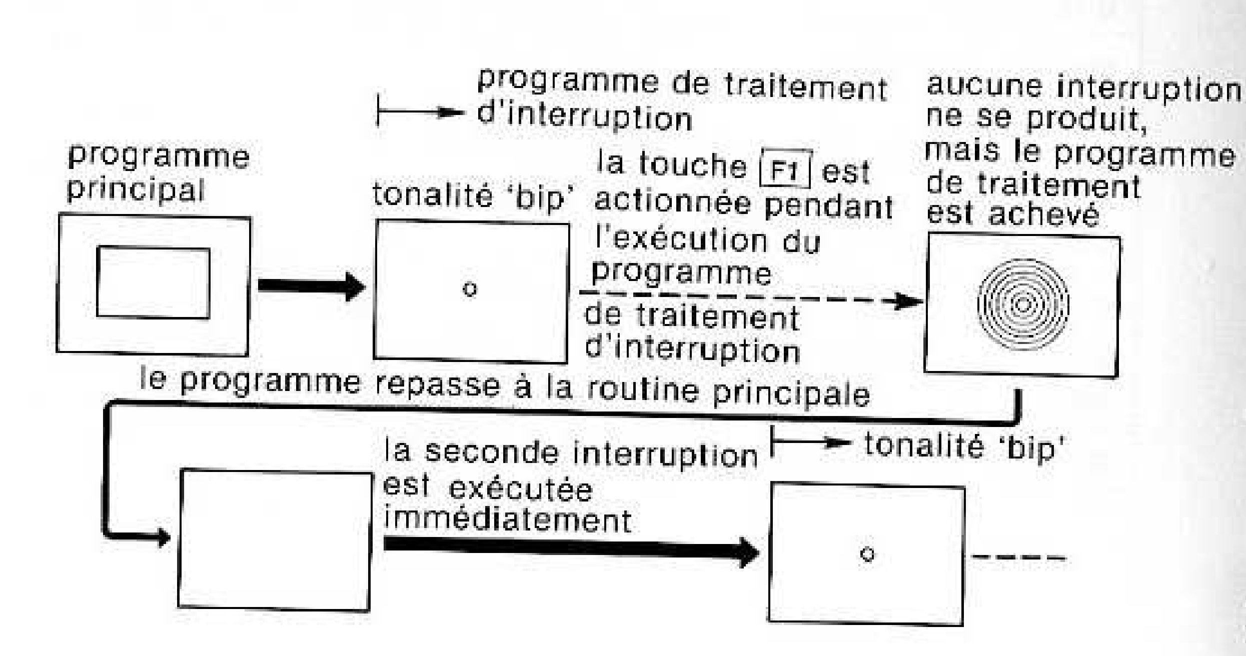
est exécuté, à la ligne 105, quand le programme de traitement d'interruption est exécuté et que ceci a pour effet d'invalider l'interruption par la touche F1.
L'énoncé KEY(N) OFF invalide une interruption par touche de fonction. N spécifie le numéro de la touche de fonction.

Le tableau suivant présente les cinq énoncés du MSX2-BASIC, permettant d'invalider des interruptions.

Interruption	Enoncé d'invalidation d'interruption
par une touche de fonction	KEY(N) OFF
par barre d'espacement, ou levier, souris, poignée ou clavier tactile	STRIG(N) OFF
par CTRL + STOP	STOP OFF
par chevauchement de lutin	SPRITE OFF
par rythmeur interne	INTERVAL OFF

MAINTIEN D'UNE INTERRUPTION

Des interruptions supplémentaires sont placées en état d'attente quand l'exécution d'un programme est aiguillée vers une routine de traitement d'interruption par une interruption. Si l'on essaie une autre interruption pendant un état d'attente, elle ne sera pas exécutée immédiatement. En revanche, le programme repassera d'abord à la routine principale par l'énoncé RETURN à la fin de la routine de traitement d'interruption. Dès le retour à la routine principale, l'énoncé— ON est automatiquement exécuté et la seconde interruption se produit alors au lieu de l'exécution de la routine principale.
Dès lors, quand on essaie une interruption au cours d'un état d'attente, cette interruption est mémorisée par l'ordinateur et elle n'est pas exécutée avant que ne soit achevée la routine de traitement de l'interruption initiale.
Dans le programme de la page 215, neuf cercles avaient été tracés par la routine de traitement d'interruption quand la touche [F1] avait été actionnée. Si la touche [F1] était actionnée à nouveau avant que ne soit tracé le dernier cercle, cette seconde interruption ne serait pas exécutée immédiatement; elle le serait immédiatement après le retour au programme principal, c'est-à-dire après le tracé du dernier cercle. Dans ce cas, au lieu de tracer un rectangle, le programme tracerait à nouveau des cercles.

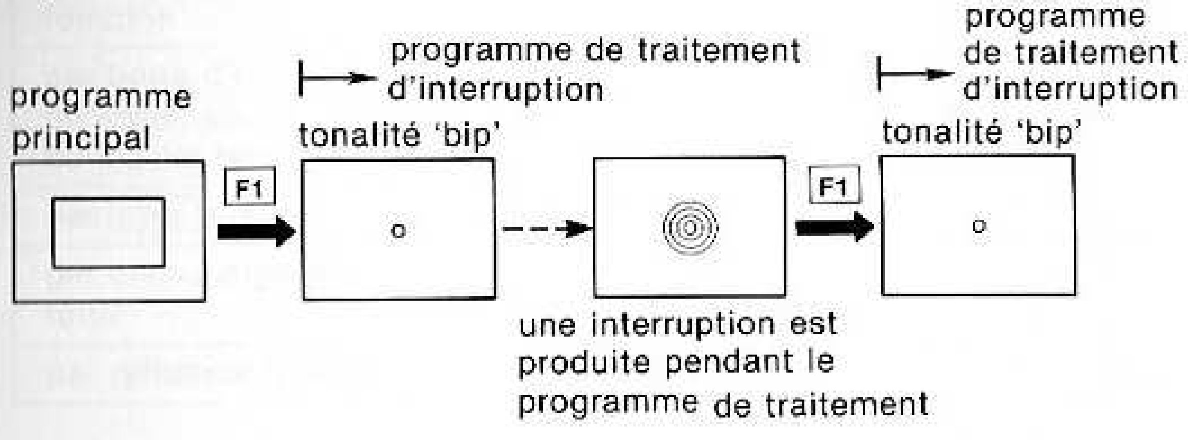


REVALIDATION D'UNE INTERRUPTION PENDANT UN PROGRAMME DE TRAITEMENT D'INTERRUPTION KEY(N) ON

Un énoncé tel que KEY(1) ON peut être inséré dans une routine de traitement d'interruption afin de revalider l'interruption lorsque la routine de traitement est exécutée. Ensuite, si l'interruption est à nouveau appliquée après qu'a commencé la routine de traitement, celle-ci sera exécutée une nouvelle fois depuis le début.

```
10 ON KEY GOSUB 100
20 KEY(1) ON
30 SCREEN 2
40 LINE (50,50)-(200,150),,B
50 GOTO 40
100 'subroutine
105 KEY(1) ON
110 BEEP:CLS
120 FOR L=10 TO 90 STEP 10
130 CIRCLE (120,100),L
140 NEXT L
150 CLS
160 RETURN 40
```

KEY(1) ON a été ajouté comme ligne 105 au programme antérieur. Elle fera qu'une interruption est provoquée quand la touche F1 est actionnée alors que le sous-programme de traitement est en train de tracer des cercles. Le programme repassera immédiatement à la ligne 100 et le sous-programme de traitement sera exécuté à nouveau depuis le début.

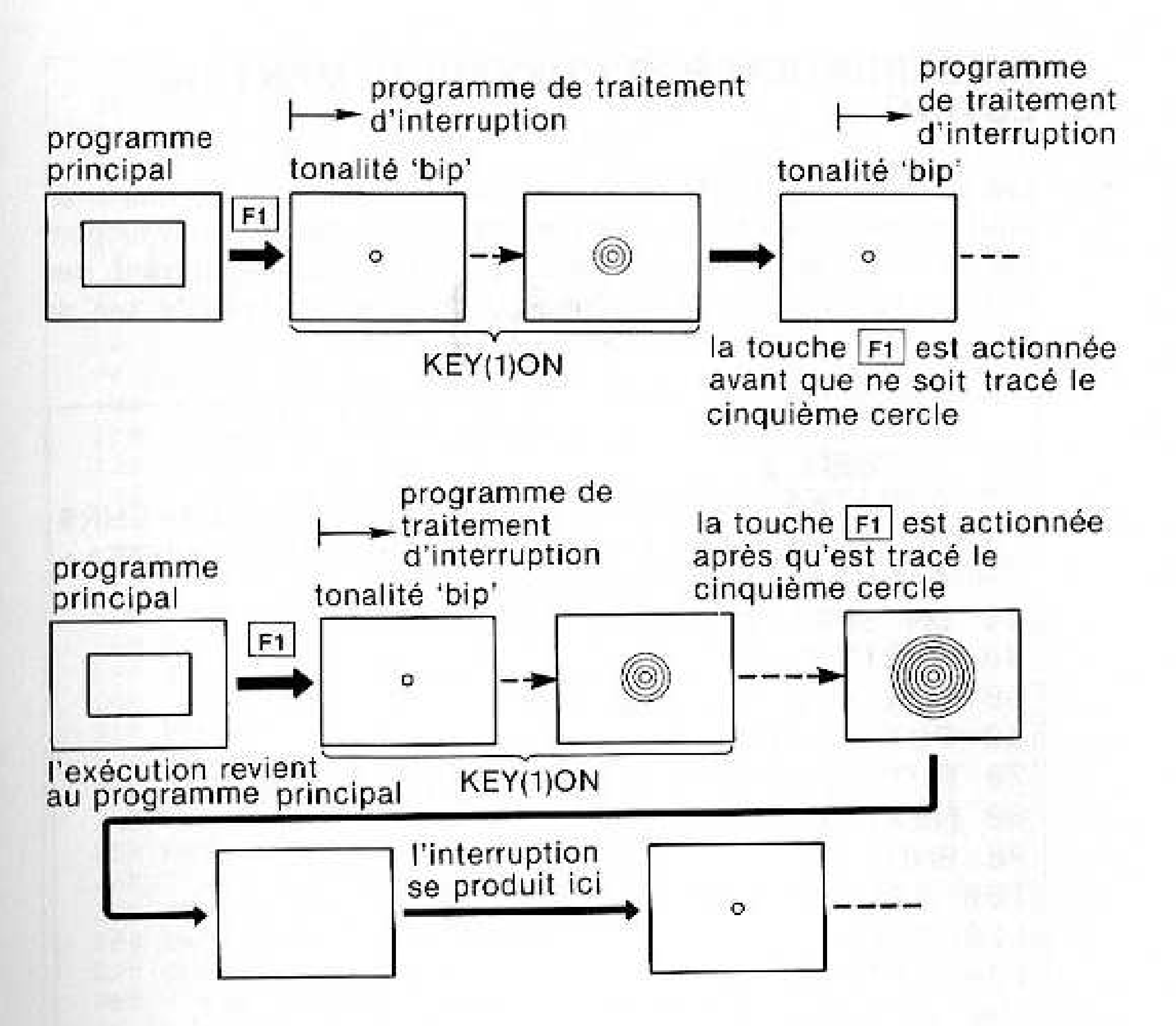


MAINTIEN D'UNE INTERRUPTION DANS UN PROGRAMME KEY(N) STOP

L'énoncé KEY(N) STOP permet de restaurer un état de maintien d'interruption après qu'une interruption a été validée par un énoncé KEY(N) ON dans un programme de traitement d'interruption.

```
10 ON KEY GOSUB 100
20 KEY(1) ON
30 SCREEN 2
40 LINE (50,50)-(200,150),,B
50 GOTO 40
100 'subroutine
105 KEY(1) ON
110 BEEP:CLS
120 FOR L=10 TO 90 STEP 10
130 CIRCLE (120,100),L
135 IF L=50 THEN KEY(1) STOP
140 NEXT L
150 CLS
160 RETURN 40
```

La ligne 135, qui exécute l'énoncé KEY(1) STOP quand la valeur de L atteint 50, a été ajoutée au programme précédent. Une autre interruption se produira immédiatement si la touche F1 est actionnée avant que le cinquième cercle ne soit tracé. Mais l'état de maintien d'interruption est restauré par la ligne 135 après que le cinquième cercle est tracé et, par conséquent, une interruption ne se produira pas immédiatement quand sera actionnée la touche F1.



L'énoncé KEY(N) STOP place une touche de fonction en état d'attente. N spécifie le numéro de la touche de fonction.
Le tableau suivant indique les cinq énoncés de maintien d'interruption, disponibles en MSX2-BASIC.

Interruption	Enoncé de maintien d'interruption
par une touche de fonction	KEY(N) STOP
par barre d'espacement, ou levier, souris, poignée ou clavier tactile	STRIG(N) STOP
par CTRL + STOP	STOP STOP
par chevauchement de lutin	SPRITE STOP
par rythmeur interne	INTERVAL STOP

INTERRUPTION PAR CHEVAUCHEMENT DE LUTINS

Les énoncés ON SPRITE GOSUB et SPRITE ON entraînent une interruption quand deux ou plusieurs motifs sprite (lutins) se chevauchent par au moins un de leurs points. Dans le programme suivant, des OVNI se déplacent de la gauche et la droite et une tonalité 'bip' se produit quand deux se chevauchent.

```
10 SCREEN 2
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 ON SPRITE GOSUB 100
40 SPRITE ON
50 FOR X=0 TO 255
60 PUT SPRITE 0,(X,100),15,0
70 PUT SPRITE 1,(255-X,100),10,0
80 NEXT X
90 END
100 SPRITE OFF
110 BEEP
120 SPRITE ON
130 RETURN
```

Exemple de programme

Le programme suivant est un jeu de tir à l'arc qui provoque des interruptions par la touche F1 et par chevauchement de lutins.
Une poussée sur la touche F1 décoche une flèche et des points sont accordés en fonction de la partie de la cible qui est touchée. Le joueur peut lancer cinq flèches et il peut rejouer quand son total atteint 1000 points.

```
10 SCREEN 5,1
20 OPEN "GRP:" FOR OUTPUT AS #1
30 ON KEY GOSUB 250 : ON SPRITE GOSUB 27
0
40 ' *** sprite definition ***
50 RESTORE 430:SN=0:GOSUB 310
60 RESTORE 520:SN=1:GOSUB 310
70 ' *** main routine ***
80 X=230:S=0:W=50
90 FOR L=1 TO 5
100  SPRITE  ON:KEY (1) ON
110  U=W:M=0:B=0:V=106:GOSUB 210
120  FOR Y=0 TO 211
130   PUT SPRITE 0,(X,Y),,0
140   U=U+B:IF M=1 THEN V=V+1
150   PUT SPRITE 1,(U,V),,1
160   IF U>250 THEN U=W:B=0
170  NEXT Y
180 NEXT L
190 IF S>=1000 THEN 80 ELSE END
200 ' *** score display ***
210 PRESET (30,10)
220 PRINT #1,USING "## : ####";L,S
230 RETURN
240 ' *** shoot ***
250 KEY (1) OFF:B=3:RETURN
260 ' *** hit ***
270 SPRITE OFF:B=0:M=1
280 S=S+(8-ABS(Y-V+1))^2*10
290 GOSUB 210 : RETURN
300 ' *** sprite definition subroutine *
**
310 SP$="":SC$=""
320 FOR SL=0 TO 7
330 READ SQ$,SC:SP=0
340  FOR SJ=1 TO 8
350   SP=SP*2-(MID$(SQ$,SJ,1)="O")
360  NEXT SJ
370  SP$=SP$+CHR$(SP):SC$=SC$+CHR$(SC)
380 NEXT SL
390 SPRITE$(SN)=SP$
400 COLOR SPRITE$(SN)=SC$
410 RETURN
420 ' *** sprite data ***
430 DATA ......OO,1
440 DATA ......OO,8
450 DATA ......OO,15
460 DATA ......OO,8
470 DATA ......OO,8
480 DATA ......OO,15
490 DATA ......OO,8
500 DATA ......OO,1
```

Lignes 50–60 : définition du lutin (sprite)

Lignes 80–180 : affiche la flèche et déplace la cible du haut vers le bas.

Lignes 210–230 : sous-programme d'affichage des points

Ligne 250 : 3 est affecté à B quand F1 est actionnée (quand une flèche est tirée)

Lignes 270–290 : calcule les points quand les lutins se chevauchent

Lignes 310–410 : sous-programme de définition du lutin

Lignes 430–500 : donnée de forme du lutin (cible)

```
510 ' *** sprite data ***
520 DATA ........,0
530 DATA ........,0
540 DATA ........,0
550 DATA OOOOOOOO,15
560 DATA ........,0
570 DATA ........,0
580 DATA ........,0
590 DATA ........,0
```

dornée de forme du lutin (flèche)

L'énoncé OPEN de la ligne 20 est nécessaire pour afficher les caractères sur l'écran en mode graphique. Ce point sera expliqué dans le Chapitre 9.

Chapitre 9
Traitement des fichiers

LES FICHIERS ET LES DISPOSITIFS DE FICHIERS

- En quoi consistent les fichiers?
- Périphériques pour fichiers
- Utilisation des fichiers programme
- Gestion des fichiers

LES FICHIERS ET LES NOMS DE FICHIERS

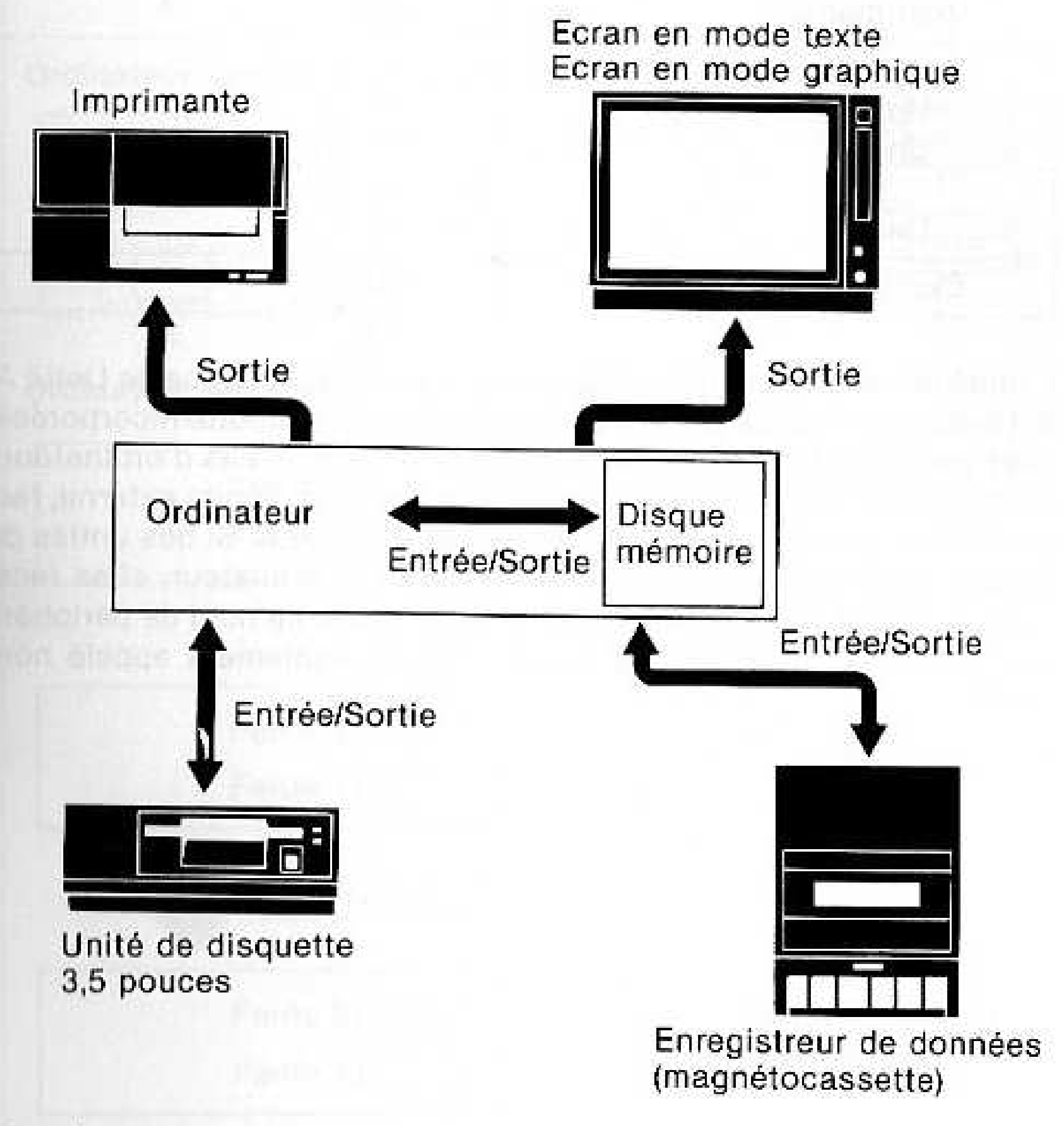
Tout ordinateur agit avec des organes périphériques qui lui sont raccordés pour utiliser des programmes ou des données, contenues dans un programme. Cette interaction pourrait être comparée à la tenue d'un journal. Imaginons que vous ayez plusieurs bibliothèques dans votre bureau. Dans l'une d'elles se trouve un cahier, portant le titre "Journal". Si vous désirez lire ce journal ou y ajouter quelque chose, vous devrez tout d'abord vous approcher de la bibliothèque en question, y chercher le cahier intitulé "Journal", puis le retirer du rayon de la bibliothèque.

L'ordinateur procède d'une manière assez semblable. Bien entendu, il n'utilise pas de cahiers, mais fait appel à des fichiers. Un fichier diffère d'un cahier en ceci qu'il n'est pas possible de le prendre en mains et de le toucher. On devrait plutôt imaginer un fichier comme une "zone" ou un certain espace, réservé sur une cassette ou une disquette.

Noms de fichiers
Le titre "Journal" a pour but de distinguer le cahier, utilisé comme journal, de tous les autres cahiers de votre bibliothèque. De la même façon, vous attribuerez des noms aux fichiers que vous utiliserez avec votre ordinateur et c'est ce que l'on appelle un **nom de fichier**.

PERIPHERIQUES POUR FICHIERS ET NOMS DE PERIPHERIQUES

Votre bibliothèque correspond au périphérique de fichiers, raccordé à l'ordinateur. Il en existe de deux types, à savoir les dispositifs d'entrée/sortie (magnétocassette, unité de disquette, etc.) et ceux qui ne permettent que la sortie des données (imprimante, moniteur vidéo, etc.). Le MSX2-BASIC possède des instructions et des énoncés spéciaux qui lui permettent d'utiliser les fichiers avec les divers périphériques qui lui sont raccordés. Le schéma suivant présente les divers dispositifs ou périphériques de fichiers, utilisables avec le MSX2-BASIC.



Des disquettes peuvent être utilisées soit dans l'unité incorporée à l'ordinateur, soit dans une unité séparée qui lui est raccordée.

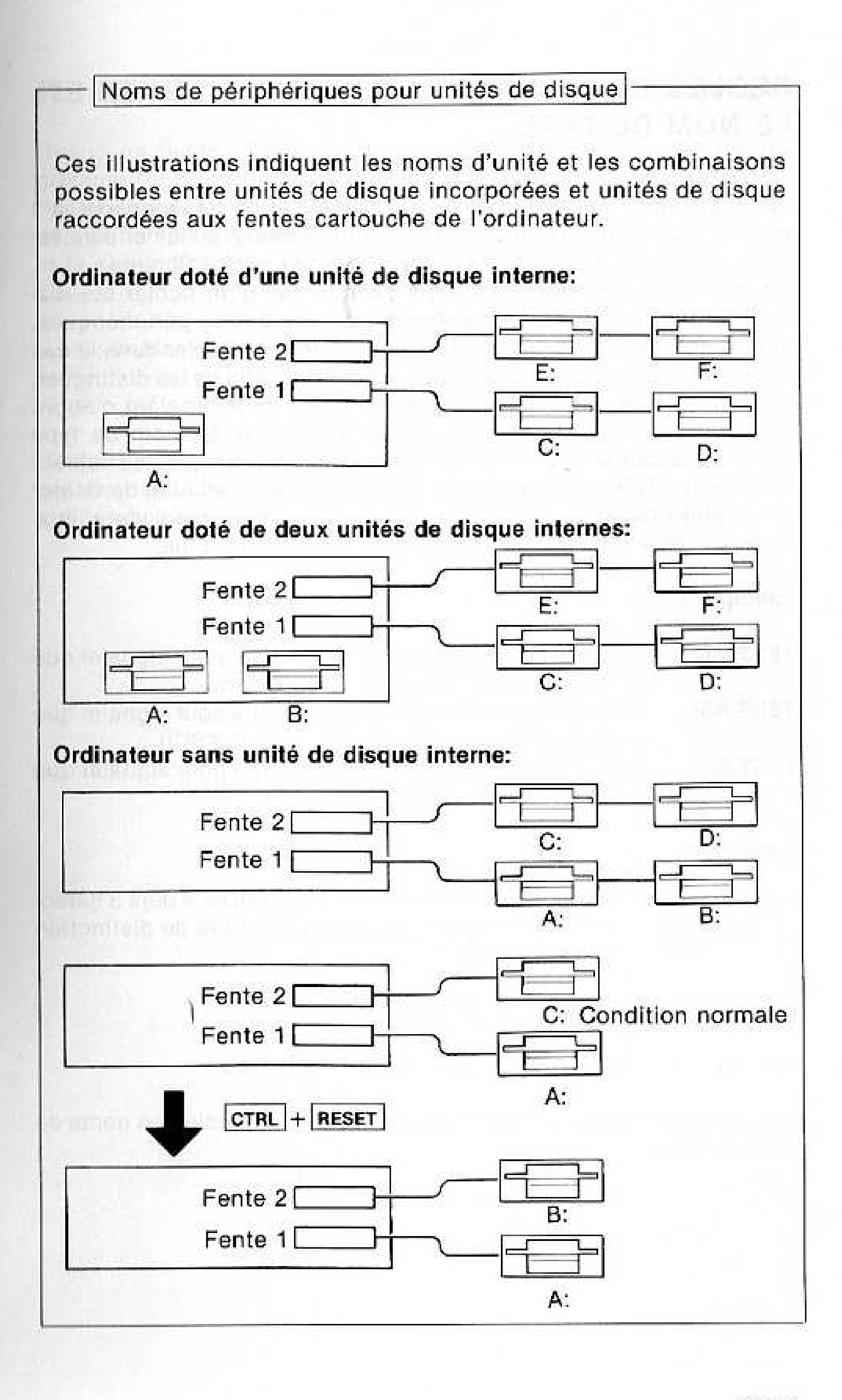
Noms de périphériques
Pour que des programmes ou des informations puissent être écrits ou consultés sur des fichiers, le périphérique où se trouvent ces derniers doit être spécifié. A cet effet, chaque type de dispositif reçoit un nom de périphérique, comme le montre le tableau suivant.

Périphérique de fichiers	Nom de périphérique
Enregistreur (magnéto-cassette)	CAS:
Ecran en mode texte	CRT:
Ecran en mode graphique	GRP:
Imprimante	LPT:
Disquette Unité A Unité B ⋮ Unité H	 A: B: ⋮ H: } nom d'unité
Disque mémoire	MEM:

L'unité de disquette, incorporée à un ordinateur, est appelée Unité A. Si l'ordinateur utilisé comporte deux unités de disquette incorporées, elles porteront le nom d'Unité A et Unité B. Dans le cas d'ordinateurs ne disposant pas d'unité de disquette incorporée, l'unité externe, raccordée sur sa fente cartouche, est appelée Unité A. Si des unités de disque complémentaires sont raccordées à l'ordinateur, elles recevront le nom d'unité B, unité C et ainsi de suite. Le nom de périphérique, donné à un disque (tel que A:, B:) est également appelé nom d'unité.

Noms de périphériques pour unités de disque

Ces illustrations indiquent les noms d'unité et les combinaisons possibles entre unités de disque incorporées et unités de disque raccordées aux fentes cartouche de l'ordinateur.



REGLES CONCERNANT LE NOM DE FICHIER ET LE NOM DE TYPE

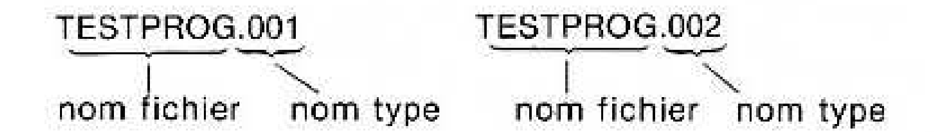
Tout nom de fichier doit commencer par une lettre de l'alphabet et il peut comporter jusqu'à 6 caractères dans le cas d'un fichier sur cassette et jusqu'à 8 caractères dans le cas des autres fichiers.
Un nom de fichier peut être omis dans le cas d'un fichier sur disquette. Ceci est valable également pour les autres périphériques, mais il est vivement conseillé d'utiliser le nom de fichier dans le cas de fichiers sur cassette ou sur disque mémoire, afin de les distinguer.
Un nom de type est ajouté au nom de fichier en intercalant d'abord un point (.), suivi de trois lettres au maximum. Le nom de type s'avère pratique pour distinguer différents types, tels que des fichiers BASIC ou ASCII, ou pour ajouter des caractères à un nom de fichier si les huit utilisables ne suffisent pas. Les exemples suivants illustrent les deux utilisations principales des noms de type.

Exemple 1

TEST.BAS	.BAS a été ajouté comme nom de type pour signaler que ce fichier est un programme en BASIC.
TEST.ASC	.ASC a été ajouté comme nom de type pour signaler que le fichier a été sauvegardé en format ASCII.
TEST.DAT	.DAT a été ajouté comme nom de type pour signaler que le fichier est un fichier de donnée.

Exemple 2

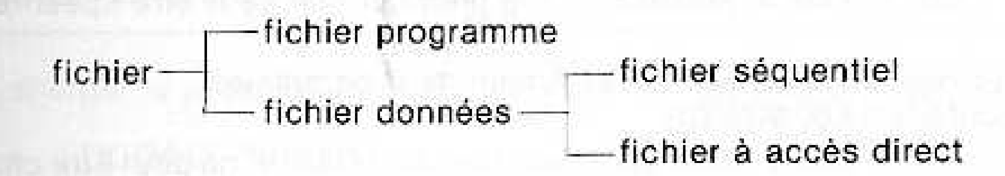
TESTPROG1
TESTPROG2 } Comme le nom de fichier TESTPROG a déjà 8 caractères, l'ordinateur ne pourra pas faire de distinction entre les deux.



Des noms de type peuvent être ajoutés pour différencier les noms de deux fichiers.

FICHIERS PROGRAMME ET FICHIERS DONNEES

Quand un fichier renferme un programme, on parle de fichier programme tandis qu'un fichier, comportant des données, est appelé fichier données. Les fichiers données se divisent en fichiers séquentiels et fichiers à accès direct selon la méthode utilisée pour l'entrée et la sortie (l'écriture et la lecture) des données.



Ci-après, nous expliquerons toutes les instructions utilisées pour la gestion des fichiers programmes sur les divers périphériques, tandis que les fichiers données seront traités dans la section suivante.

UTILISATION DES FICHIERS PROGRAMME

Les instructions suivantes servent à sauvegarder ou à charger des programmes, ou encore à fusionner des programmes dans la mémoire.

CSAVE, CLOAD	pour cassette uniquement
SAVE, LOAD, MERGE........	le périphérique peut être spécifié

Sauvegarde, chargement et fusion de programmes
SAVE, LOAD, MERGE

Tout programme sauvegardé par l'énoncé CSAVE ne peut être chargé que par l'énoncé CLOAD, tandis que l'énoncé LOAD est requis pour le chargement d'un programme sauvegardé par l'énoncé SAVE. Le nom de fichier, utilisé pour la sauvegarde d'un programme, doit être spécifié pour charger ce programme.
L'énoncé MERGE charge un programme et il le fusionne à un autre, déjà placé en mémoire. Seuls des programmes sauvegardés en format ASCII peuvent être fusionnés.

Instruction de sauvegarde	Format de sauvegarde	Instruction de chargement
CSAVE	Langage intermédiaire	CLOAD
SAVE (cassette ou disque mémoire)	ASCII Format	LOAD ou MERGE
SAVE (disquette)	Langage intermédiaire	LOAD
SAVE, A (disquette seulement)	Format ASCII	LOAD ou MERGE

Dans le cas des cassettes, l'instruction CSAVE sauvegarde un programme en langage intermédiaire, tandis que l'instruction SAVE le sauvegarde en format ASCII. Seule l'instruction SAVE est utilisable avec une disquette. Si l'option A est ajoutée à l'instruction SAVE d'une disquette, le programme sera sauvegardé en format ASCII. Si elle est omise, il le sera en langage intermédiaire.
Seule l'instruction SAVE est utilisable pour le disque mémoire également. Tout programme sauvegardé sur le disque mémoire le sera en format ASCII.

Exploitation d'un fichier programme en utilisant une cassette
CSAVE, CLOAD

Un programme placé en mémoire peut être sauvegardé sur une cassette par une des deux méthodes suivantes:

```
CSAVE "PROG"——  sauvegardé en langage intermédiaire
                sous le nom de fichier PROG—①
SAVE "CAS:PROG"—  sauvegardé en format ASCII sous le nom
                  de fichier PROG—②
```

Pour charger le programme sauvegardé en ① ci-dessus, exécutez:

```
CLOAD "PROG"
```

Pour charger le programme sauvegardé en ② ci-dessus, exécutez:

```
LOAD "CAS:PROG"
```

Pour fusionner le programme sauvegardé en ② ci-dessus à un autre programme placé en mémoire, exécutez:

```
MERGE "CAS:PROG"
```

Exploitation d'un fichier programme en utilisant une disquette
SAVE, LOAD

Un programme placé en mémoire peut être sauvegardé sur un disque (unité A) par une des deux méthodes suivantes:

```
SAVE "A:PROG.BAS"——Sauvegardé en langage
                   intermédiaire avec le nom
                   de fichier/type PROG.BAS—①
SAVE "A:PROG.ASC",A——Sauvegardé en format
                     ASCII avec le nom de
                     fichier/type PROG.BAS—②
```

Les programmes sauvegardés en ① et ② ci-dessus peuvent être chargés en exécutant:

```
LOAD "A:PROG.BAS"

LOAD "A:PROG.ASC"
```

Pour fusionner le programme sauvegardé en ② ci-dessus à un autre programme placé en mémoire, exécutez:

```
MERGE "A:PROG.ASC"
```

Si l'on n'utilise qu'une seule unité de disque, le nom d'unité (A:) peut être omis. Si ce nom est omis alors que deux ou plusieurs unités de disque sont en service, c'est l'unité actuellement utilisée qui est choisie.

Exploitation d'un fichier programme en utilisant le disque mémoire
En MSX2-BASIC, une partie de la mémoire interne, distincte de celle qui sert à stocker les programmes, peut servir pour sauvegarder temporairement un programme. Comme cette portion de la mémoire est utilisée comme une disquette, on parle de **fonction de disque mémoire.** En revanche, la partie de la mémoire qui sert à conserver les programmes est appelée zone programme. Tout programme, stocké dans la zone mémoire peut être affiché sur l'écran par une instruction LIST et exécuté par une instruction RUN. Si un programme, stocké dans la zone mémoire, est sauvegardé sur le disque mémoire, cette zone programme pourra être utilisée pour la rédaction d'un autre programme.
Cependant, n'oubliez pas que tout programme placé sur le disque mémoire est effacé dès l'instant où l'ordinateur est mis hors tension.

Avant son utilisation, le disque mémoire doit être initialisé par l'énoncé CALL MEMINI (initialisation mémoire).

CALL MEMINI [(taille)]

La portion de la mémoire qui sera utilisée par le disque mémoire est spécifiée par la "taille" de l'énoncé CALL MEMINI. Cette taille est spécifiée en octets et elle peut aller de 1023 à 32767 octets. Le réglage implicite est de 32767 octets quand cette spécification est omise.
Lorsque l'énoncé CALL MEMINI est exécuté, l'écran affiche la taille spécifiée pour le disque mémoire.

```
CALL MEMINI
32000 bytes allocated
Ok
```

Une fois que l'énoncé CALL MEMINI a été exécuté, un programme placé dans la zone programme peut être sauvegardé sur le disque mémoire par l'instruction suivante:

```
SAVE "MEM:PROG.BAS"
```
— sauvegardé en format ASCII sous le nom de fichier/type PROG.BAS

Pour charger un programme issu du disque mémoire vers la zone programme, exécutez l'énoncé suivant:

```
LOAD "MEM:PROG.BAS"
```

Pour fusionner un programme issu du disque mémoire avec un programme se trouvant dans la zone programme, exécutez:

```
MERGE "MEM:PROG.BAS"
```

Chargement et exécution d'un programme RUN
L'instruction RUN, suivie du nom de fichier/type, est utilisée pour charger un programme issu d'une disquette ou du disque mémoire et pour l'exécuter immédiatement.

RUN "[nom périphérique] nom fichier [.nom type]"

```
RUN "PROG.BAS"
```
— "PROG.BAS" de l'unité A est chargé et exécuté

```
RUN "MEM:PROG.BAS"
```
— "PROG.BAS" du disque mémoire est chargé et exécuté

GESTION DE FICHIERS

Affichage des noms de fichier FILES, CALL MFILES
Pour afficher les noms des fichiers se trouvant sur la disquette, exécutez:

```
FILES
```

Le nom de fichier/type est spécifié pour vérifier si un fichier particulier se trouve sur un disque.

```
FILES "PROG.BAS"
```

Pour afficher les noms des fichiers se trouvant sur le disque mémoire, exécutez:

```
CALL MFILES
```

CALL MFILES

Le nombre d'octets restants dans la mémoire du disque est également affiché après l'exécution de l'instruction CALL MFILES.

Effacement de fichiers KILL, CALL MKILL
L'instruction KILL s'emploie pour effacer un fichier se trouvant sur une disquette. Par exemple, pour effacer un fichier, portant le nom "TEST.DAT", on exécutera:

```
KILL "TEST.DAT"
```

Par contre, CALL MKILL s'emploie pour effacer un fichier, placé sur le disque mémoire. Pour effacer un fichier, portant le nom "PROG.BAS", on exécutera:

```
CALL MKILL ("PROG.BAS")
```

CALL MKILL ("nom fichier [.nom type]")

Changement d'un nom de fichier NAME, CALL MNAME
L'instruction NAME s'emploie pour changer le nom de fichier et/ou le nom de type d'un fichier, se trouvant sur disquette.

NAME "[nom unité] ancien nom fichier [.ancien nom type]" AS "nouveau nom fichier [.nouveau nom type]"

CALL MNAME s'emploie pour changer le nom de fichier et/ou le nom de type d'un fichier, se trouvant sur le disque mémoire.

CALL MNAME ("ancien nom fichier [.ancien nom type]")
AS ("nouveau nom fichier [.nouveau nom type]")

```
NAME "OLD.BAS" AS "NEW.BAS"
```

———— "OLD.BAS" de l'unité A est changé en "NEW.BAS"

```
CALL MNAME ("OLD.BAS" AS "NEW.BAS")
```

———— "OLD.BAS" du disque mémoire est changé en "NEW.BAS"

FICHIER PROGRAMME A MISE EN MARCHE AUTOMATIQUE

Un programme à mise en marche automatique sera chargé depuis le disque et il sera automatiquement exécuté par la mise sous tension de l'ordinateur ou par une poussée sur la touche RESET.
Pour créer un programme à mise en marche automatique, spécifiez

AUTOEXEC.BAS

comme nom/type de fichier du programme lorsqu'il est sauvegardé sur un disque. Un seul programme à mise en marche automatique peut être créé par disque.

UTILISATION D'UN FICHIER SEQUENTIEL

- Qu'est-ce qu'un fichier séquentiel?
- Ecriture de données dans un fichier séquentiel
- Lecture de données issues d'un fichier séquentiel
- Addition de données
- Ecriture de caractères sur un écran graphique
- Nombre de fichiers ouvrables en une fois

FICHIERS SEQUENTIELS ET FICHIERS A ACCES DIRECT

Des fichiers données servent à manipuler des informations qui sont traitées par un programme rédigé en BASIC. Par exemple, dans le cas d'un programme "Annuaire téléphonique", le programme proprement dit est sauvegardé comme un fichier programme, tandis que les informations, concernant les noms et les numéros de téléphone des personnes répertoriées et traitées par le programme, sont sauvegardées comme un fichier données.

Il existe deux types de fichiers données, à savoir 1) le fichier séquentiel, servant à écrire et lire les données en séquence à partir de son début; et 2) le fichier à accès direct, permettant d'écrire et de lire les données à un endroit déterminé du fichier. Le tableau suivant montre les périphériques utilisables pour les fichiers séquentiels et les fichiers à accès direct.

Périphériques utilisables avec des fichiers séquentiels	Périphériques utilisables avec des fichiers à accès direct
cassette disquette imprimante écran en mode texte/graphique disque mémoire	disquette

Dans cette section, on se servira, pour l'explication des fichiers séquentiels, d'une disquette placée dans l'unité A.
Les énoncés suivants servent pour l'entrée/sortie de donnnées dans un fichier séquentiel.

OPEN	ouvre un fichier
PRINT# PRINT# USING	écrit une donnée dans un fichier
INPUT# LINE INPUT#	lit une donnée contenue dans un fichier
CLOSE	referme un fichier

ECRITURE DE DONNEES DANS UN FICHIER SEQUENTIEL OPEN FOR OUTPUT

La marche à suivre pour placer une donnée dans un fichier séquentiel est la suivante:
(1) Ouvrir le fichier par un énoncé OPEN.
(2) Ecrire la donnée dans le fichier par un énoncé PRINT#.
(3) Refermer le fichier par un énoncé CLOSE.

Le format de l'énoncé OPEN pour placer la donnée est:

OPEN "[nom périphérique] [nom fichier [.nom type]]" FOR OUTPUT AS [#] numéro fichier

L'énoncé OPEN prépare un fichier, auquel la donnée sera fournie, en donnant le nom de fichier spécifié au périphérique spécifié. Quand l'ordinateur écrit une donnée dans un fichier ou qu'il lit la donnée d'un fichier, une partie de la mémoire est réservée comme registre-tampon. Celui-ci stocke temporairement la donnée jusqu'à ce qu'y soit placée la quantité voulue; ensuite, le tampon sort ou entre les données en question. En MSX2-BASIC, un maximum de 16 registres-tampon peuvent être spécifiés (7 au maximum pour une disquette). Le numéro de fichier, spécifié dans l'énoncé OPEN, détermine lequel des 16 registres-tampon sera utilisé. Le réglage implicite initial est 1.

Après l'ouverture d'un fichier par l'énoncé OPEN, l'énoncé PRINT# s'emploie pour fournir la donnée au fichier. Le format est le suivant:

PRINT# numéro fichier,[expression] [séparateur] [expression...]

Le numéro de fichier doit être le même que celui qui a été spécifié dans l'énoncé OPEN.
A chaque exécution de l'énoncé PRINT#, le code retour (&H0D) et le code interligne (&H0A) sont automatiquement ajoutés à la donnée qui a été écrite par l'énoncé PRINT#. Ces deux codes remplissent le rôle d'un signe, afin de séparer la donnée de celle qui sera écrite ensuite dans le fichier. Si les données sont de type caractère, on pourra utiliser une virgule entre parenthèses (",") pour séparer plusieurs séries de données dans un énoncé PRINT# comme ci-dessous:

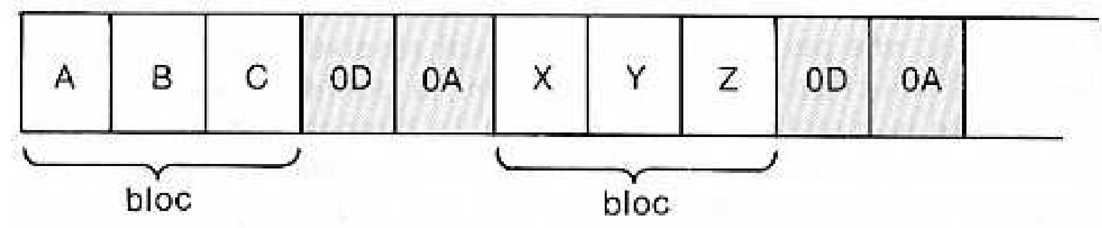
```
PRINT #1,A$;",";B$
```

Etant donné que la virgule sert à séparer des données, A$ et B$ seront traités comme des séries d'informations différentes à la lecture des données. (Si les données sont de type numérique, les séparations sont effectuées automatiquement entre chaque poste et, par conséquent, aucun signe particulier n'est requis.)

Chaque groupe de données, séparé par une paire 0D, 0A, est appelé un bloc. Par exemple, si "ABC" est affecté à la variable A$, que "XYZ" est affecté à la variable B$ et que

```
PRINT #1,A$
PRINT #1,B$
```

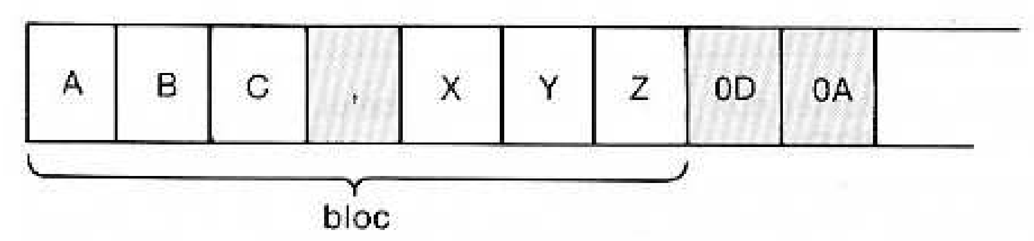
est exécuté, les données seront écrites comme suit sur le disque:



Ou bien si

```
PRINT #1,A$;",";B$
```

est exécuté, les deux variables étant écrites dans un énoncé PRINT# et étant séparées par ",", les données seront alors écrites comme suit:



Dans ce cas, les données ABC et XYZ sont écrites en un seul bloc, mais la virgule les sépare en deux séries d'informations.
Après que les données ont été sorties, le fichier est refermé par l'énoncé CLOSE.

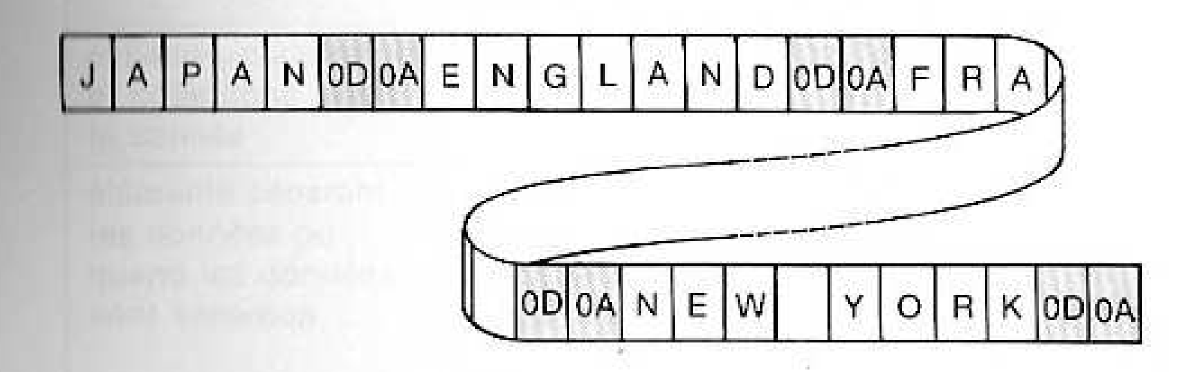
CLOSE [#] [numéro fichier]

Lorsque l'énoncé CLOSE est exécuté, le numéro de fichier n'est plus affecté à ce fichier particulier et il peut donc être utilisé pour ouvrir un autre fichier.

Rédigeons à présent un programme qui écrit des données dans un fichier séquentiel.

```
10 DIM A$(1,3)
20 OPEN "A:DATA.DAT" FOR OUTPUT AS #1
30 FOR L=0 TO 1
40 FOR M=0 TO 3
50 READ A$(L,M)
60 PRINT #1,A$(L,M)
70 NEXT M
80 NEXT L
90 CLOSE #1
100 END
110 DATA JAPAN,ENGLAND,FRANCE,U.S.A.
120 DATA TOKYO,LONDON,PARIS,NEW YORK
```

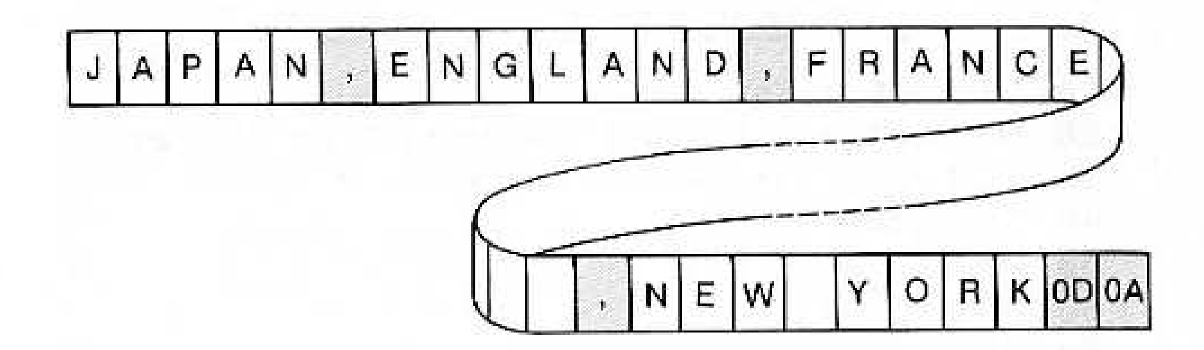
Quand ce programme est exécuté, un fichier de données, portant le nom "DATA.DAT", est créé sur la disquette de l'unité A. La donnée de type caractère "JAPAN" est d'abord écrite dans ce fichier, suivie d'une paire 0D, 0A puis de ENGLAND et d'autres noms de pays ou de villes y sont écrits en séquence, séparés chaque fois par une paire 0D, 0A.



Si la ligne 60 est changée en

```
PRINT #1,A$(L,M);",";
```

les données seront écrites comme suit:



LECTURE DE DONNEES ISSUES D'UN FICHIER SEQUENTIEL OPEN FOR INPUT

La marche à suivre pour entrer des données provenant d'un fichier séquentiel est la suivante:
(1) Ouvrir un fichier par un énoncé OPEN.
(2) Lire la donnée, issue du fichier par un énoncé INPUT# ou un énoncé LINE INPUT#.
(3) Refermer le fichier par un énoncé CLOSE.

Le format de l'énoncé OPEN est le suivant:

OPEN "[nom périphérique] [nom fichier [.nom type]]"
FOR INPUT AS [#] numéro fichier

L'énoncé OPEN ouvre un fichier pour y entrer des données. Sauf spécification contraire dans le cas de l'énoncé MAXFILES (voir en page 251), seul le numéro "1" peut servir comme numéro de fichier.
Une fois que le fichier est ouvert, l'énoncé INPUT # est utilisé pour lire la donnée.

INPUT[#] numéro fichier, variable

L'énoncé INPUT# affecte une série de données à la variable.
Le tableau suivant indique comment sont lues les données par l'énoncé INPUT#.

	Données de type numérique	Données de type caractère
espaces, code retour et code ligne avant la donnée	ignoré	ignoré
éléments séparant les données ou quand les données sont séparées	espace virgule code retour code interligne	virgule code retour code interligne quand 255 caractères ont été entrés
quand la donnée est entre " "	—	les données entre " " sont lues comme une série de données

L'énoncé LINE INPUT # est employé pour lire les données de type caractère et il entre uniquement le code retour comme séparateur de données.
Le fichier est refermé par l'énoncé CLOSE après l'entrée des données et le numéro de fichier n'a plus aucun rapport avec le fichier antérieur.

Rédigeons à présent un programme qui lira et affichera sur l'écran les données, issues du fichier "DATA.DAT", réalisé précédemment.

```
10 DIM A$(1,3)
20 OPEN "A:DATA.DAT" FOR INPUT AS #1
30 FOR L=0 TO 1
40 FOR M=0 TO 3
50 INPUT #1,A$(L,M)
60 NEXT M
70 NEXT L
80 CLOSE #1
90 FOR M=0 TO 3
100 PRINT A$(0,M),A$(1,M)
110 NEXT M
```

L'énoncé INPUT # de la ligne 50 lit les données en séquence dans la variable en tableau A$, tandis que les lignes 90 à 110 affichent ces données sur l'écran.

Observons ce qui se passe si nous essayons de lire les données, provenant du fichier "DATA.DAT" en utilisant le programme suivant.

```
10 OPEN "A:DATA.DAT" FOR INPUT AS #1
20 INPUT #1,A$
30 PRINT A$
40 GOTO 20
```

Lorsque ce programme est exécuté, JAPAN, ENGLAND et les autres données sont affichées sur l'écran. Cependant, même après la lecture de la dernière donnée NEW YORK, le programme essaie encore d'entrer davantage de données. Dans ce cas, après que le fichier est terminé, le message d'erreur

Input past end

est affiché. Pour éviter que ceci se produise, on pourra utiliser la fonction EOF.

EOF (numéro de fichier)

```
10 OPEN "A:DATA.DAT" FOR INPUT AS #1
15 IF EOF(1)=-1 THEN GOTO 50
20 INPUT #1,A$
30 PRINT A$
40 GOTO 15
50 CLOSE #1
```

La fonction EOF rend −1 quand la dernière donnée d'un fichier a été lue. Dans le programme précédent, cette fonction est utilisée chaque fois qu'une donnée est lue, afin de vérifier s'il existe d'autres données dans ce fichier.

ADDITION DE DONNEES OPEN FOR APPEND

L'addition de données à un fichier préalablement réalisé est possible uniquement quand on emploie un fichier séquentiel sur une disquette ou sur le disque mémoire. Pour ajouter des données, le fichier doit d'abord être ouvert par un énoncé OPEN.

OPEN "[nom périphérique] [nom fichier [.nom type]]" FOR APPEND AS [#] numéro fichier

Les trois programmes ci-après servent à écrire, à lire et à ajouter des données sur une disquette.

Ecriture de données

```
10 OPEN "A:TEST.DAT" FOR OUTPUT AS #1
20 FOR L=1 TO 3
30 READ A$
40 PRINT #1,A$
50 NEXT L
60 CLOSE #1
70 END
80 DATA JAPAN,ENGLAND,FRANCE
```

Ce programme crée un fichier, intitulé "TEST.DAT" sur la disquette de l'unité A et il y écrit trois blocs d'informations: JAPAN, ENGLAND, FRANCE.

Lecture de données

```
10 OPEN "A:TEST.DAT" FOR INPUT AS #1
20 IF EOF(1)=-1 GOTO 60
30 INPUT #1,A$
40 PRINT A$
50 GOTO 20
60 CLOSE #1
70 END
```

Les données, écrites dans le programme d'écriture, sont lues, affectées à la variable A$ et affichées sur l'écran.

Addition de données

```
10 OPEN "A:TEST.DAT" FOR APPEND AS #1
20 FOR L=1 TO 2
30 READ A$
40 PRINT #1,A$
50 NEXT L
60 CLOSE #1
70 END
80 DATA U.S.A.,CHINA
```

Le programme écriture précédant a déjà servi à écrire trois blocs d'information dans le fichier "TEST.DAT", ouvert à la ligne 10 de ce programme. Cependant, comme FOR APPEND est spécifié dans l'énoncé OPEN, la donnée écrite par l'énoncé PRINT# à la ligne 40 est ajoutée après les trois premiers blocs d'information. Par conséquent, après exécution de ce programme, le fichier "TEST.DAT" contiendra cinq blocs d'information, à savoir JAPAN, ENGLAND, FRANCE, U.S.A., CHINA.
Si FOR OUTPUT a été utilisé au lieu de FOR APPEND, les données ajoutées seront écrites au début du fichier et les trois premiers blocs d'information seront effacés.

ECRITURE DE CARACTERES SUR UN ECRAN GRAPHIQUE

Il n'est pas possible d'afficher des caractères sur un écran en mode graphique en se servant de l'énoncé PRINT. Pour afficher des caractères sur un écran graphique, celui-ci est considéré comme un périphérique pour fichiers et les caractères qui doivent y être affichés lui sont fournis comme un fichier séquentiel.

```
10 SCREEN 2
20 OPEN "GRP:" FOR OUTPUT AS #1
30 PRINT #1,"How do you do?"
40 GOTO 40
```

Lorsque le programme ci-dessus est exécuté, l'écran passe en mode graphique el il affiche "How do you do?".

Pour spécifier un emplacement d'affichage, une instruction graphique est exécutée immédiatement avant l'énoncé PRINT#. L'emplacement spécifié par cette instruction devient le coin supérieur gauche de la matrice 8×6 points du premier caractère dans la chaîne de caractères de l'énoncé PRINT#.

```
10 SCREEN 2
20 OPEN "GRP:" FOR OUTPUT AS #1
25 PRESET (100,50)
30 PRINT #1,"How do you do?"
40 GOTO 40
```

L'emplacement (100,50), utilisé dans l'instruction PRESET à la ligne 25 de ce programme, devient le coin supérieur gauche de la chaîne de caractères, fournie par la ligne 30.

NOMBRE DE FICHIERS OUVRABLES EN UNE FOIS
MAXFILES

Seul le numéro "1" peut être spécifié comme numéro de fichier en MSX2-BASIC lors de son initialisation. Ceci signifie que, dans un programme, un seul fichier pourra être ouvert à un moment donné. Si vous désirez ouvrir plusieurs fichiers en même temps, vous devrez spécifier ce nombre au préalable en utilisant

MAXFILES = expression

Par exemple, si vous spécifiez

```
MAXFILES=5
```

un maximum de 6 fichiers, de 0 à 5, pourront alors être ouverts simultanément. Les numéros de fichiers que l'on peut spécifier vont de 0 à 15 (ces numéros étant limités de 0 à 6 dans le cas d'une disquette.) Comme le numéro de fichier 0 est réservé pour les instructions CSAVE, CLOAD, CLOAD?, LOAD et SAVE, si l'on exécute

```
MAXFILES=0
```

seules les instructions CSAVE, CLOAD, CLOAD?, LOAD et SAVE pourront être utilisées.
L'énoncé MAXFILES doit être exécuté soit au début d'un programme, soit en mode direct.

UTILISATION D'UN FICHIER A ACCES DIRECT

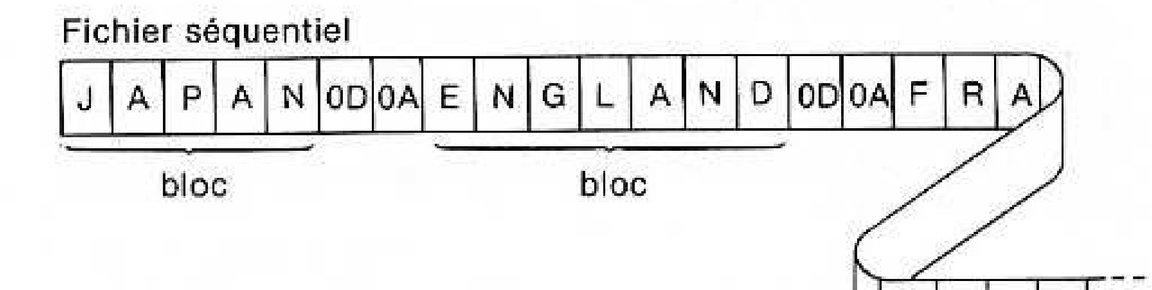
- En quoi consiste un fichier à accès direct?
- Ecriture de données dans un fichier à accès direct
- Lecture de données issues d'un fichier à accès direct

EN QUOI CONSISTE UN FICHIER A ACCES DIRECT?

Comparaison avec un fichier séquentiel

Un fichier à accès direct ne peut être utilisé que sur une disquette. Les illustrations suivantes montrent comment des données sont écrites respectivement sur un fichier séquentiel et sur un fichier à accès direct.

Fichier séquentiel



Fichier à accès direct

bloc 1	J	A	P	A	N								T	O	K	Y	O					
bloc 2	E	N	G	L	A	N	D						L	O	N	D	O	N				
bloc 3	F	R	A	N	C	E							P	A	R	I	S					
bloc 4																						

22 octets

Comme le montre le schéma ci-dessus, un fichier séquentiel ressemble à une suite de données, écrites dans l'ordre sur une bande de papier. Un bloc est une unité pour l'écriture sur le fichier à la fois, la longueur de ce bloc dépendant de la quantité des données.
Par contre, un fichier à accès direct pourrait être comparé à une série de morceaux de papier, coupés à la même longueur; chacun d'eux représente un bloc séparé, portant son propre numéro, tel que 1, 2, 3, etc. Dans l'illustration du fichier à accès direct ci-dessus, les bouts de papier avaient été coupés à une longueur, permettant de contenir 22 caractères.
Comme les blocs d'un fichier à accès direct ressemblent à des morceaux de papier séparés, il est possible d'en choisir un seul, d'y écrire des données ou de lire celles-ci qui s'y trouvent. Par exemple, si l'on choisit le bloc 3, on obtiendra de celui-ci la donnée "FRANCE PARIS". Mais l'on pourra aussi spécifier le bloc 10, sauter au-dessus des autres et y écrire une nouvelle donnée.
Un dernier point qu'il convient de mentionner, c'est que même les endroits où des données sont écrites dans chaque bloc d'un fichier à accès direct sont traités d'une façon ordonnée.

Instructions et énoncés utilisés pour opérer un fichier à accès direct
Les énoncés suivants sont employés pour effectuer des entrées/sorties vers/d'un fichier à accès direct.

OPEN .... ouvre un fichier
FIELD..................... spécifie le format d'un bloc
LSET, RSET........... écrit une donnée dans un bloc
PUT .... extrait un bloc vers un fichier
GET .... entre un bloc d'un fichier
CLOSE .... referme un fichier

ECRITURE DE DONNEES DANS UN FICHIER A ACCES DIRECT

La marche à suivre pour l'écriture de données dans un fichier à accès direct est la suivante.
(1) Ouvrir le fichier par un énoncé OPEN.
(2) Spécifier le format d'un bloc par un énoncé FIELD.
(3) Ecrire la donnée dans un bloc par les énoncés LSET, RSET.
(4) Sortir la donnée avec un énoncé PUT.
(5) Refermer le fichier.

Ci-après, nous expliquerons en détail chaque démarche de cette procédure.

(1) Ouvrir le fichier par un énoncé OPEN
Pour y écrire ou en lire des données, un fichier doit être ouvert par un énoncé OPEN. Le format de l'énoncé OPEN est le suivant:

OPEN "[nom périphérique] nom fichier [.nom type]" AS[#] numéro fichier [LEN = longueur de bloc]

Le nom de périphérique doit toujours être le nom de l'unité de disque puisqu'un fichier à accès direct ne peut être opéré que sur une disquette.

Quand l'énoncé OPEN est exécuté, un fichier, portant le nom de fichier et le nom de type spécifiés, est prêt pour la sortie de données sur l'unité de disque spécifiée. Tout comme pour les fichiers séquentiels, les numéros de fichier de 0 à 6 peuvent être spécifiés, mais à l'état d'initialisation, seul le numéro "1" est utilisable.
A la différence d'un fichier séquentiel, dans un fichier à accès direct, les données peuvent être écrites ou lues dans n'importe quelle partie. La plus petite unité, utilisable pour la lecture ou l'écriture, est appelée un bloc. La taille d'un bloc en octets est spécifiée par la "longueur de bloc" dans l'énoncé OPEN. La taille spécifiable va de 1 à 256 octets. En cas d'omission de cette spécification, la valeur de 256 octets est automatiquement spécifiée.

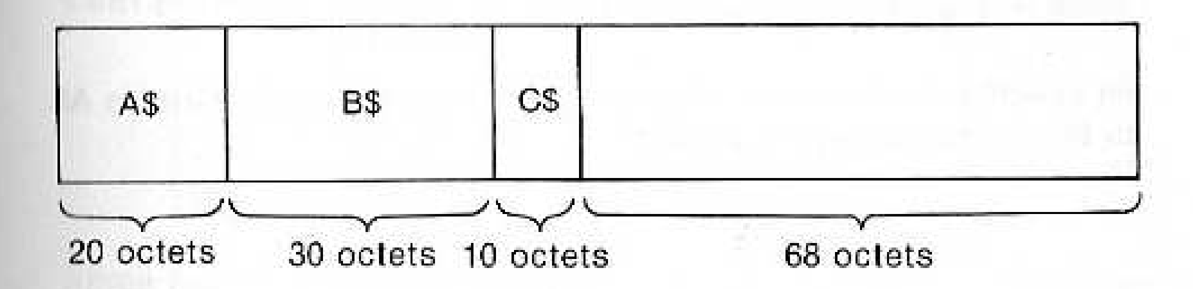
(2) Spécifier le format d'un bloc par un énoncé FIELD

FIELD[#] numéro de fichier, longueur caractère AS variable en chaîne [, longueur caractère AS variable en chaîne]...

L'énoncé FIELD spécifie les variables utilisées pour l'écriture et la lecture des données et il affecte la longueur de caractère à chaque variable dans un bloc.
Toute donnée traitée par un fichier à accès direct est une donnée de type en chaîne.

```
FIELD #1,20 AS A$,30 AS B$,10 AS C$
```

Dans l'exemple précédent, un bloc est divisé en fichier #1 pour l'entrée/sortie et 20 octets sont affectés à la variable A$, 30 octets à B$ et 10 octets à C$. Comme la longueur totale de ces variables est de 60 octets, la longueur de bloc devra, au préalable, être spécifiée comme 60 octets ou davantage dans l'énoncé OPEN précédent. Si l'on spécifie, par exemple, une longueur de 128 octets, l'énoncé FIELD divisera le bloc comme suit:



Dans cet exemple, aucune donnée n'est entrée/sortie pour les 68 octets restants et il s'agit là d'un gaspillage d'espace sur le disque. Il serait donc préférable de ne spécifier que 60 octets comme longueur de bloc dans l'énoncé OPEN.

(3) Ecrire la donnée dans le bloc par les énoncés LSET, RSET
Une fois que le bloc a été divisé et que les valeurs ont été affectées aux variables par l'énoncé FIELD, les données de sortie peuvent être placées dans le bloc. A cet effet, on se servira de l'énoncé LSET qui assure un cadrage à gauche, ou de RSET pour un cadrage à droite dans le bloc.

LSET variable en chaîne = données en chaîne
RSET variable en chaîne = données en chaîne

Les variables en chaîne de ces énoncés sont celles qui ont été préalablement affectées au bloc par l'énoncé FIELD. Les données en chaîne sont les données qui doivent être écrites dans le fichier en se servant des variables en chaîne.
Si

```
LSET A$=X$
```

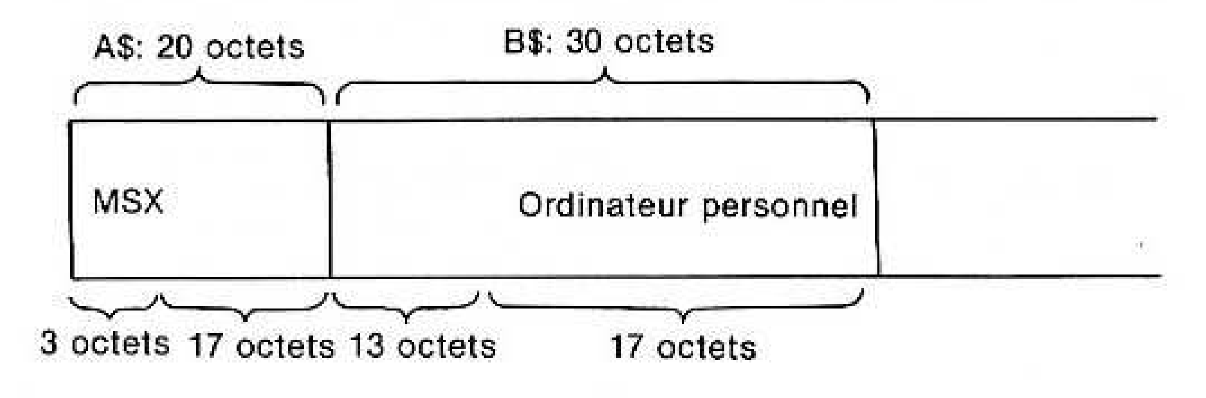
est spécifié, la donnée en chaîne X$ sera placée dans la variable A$ du bloc avec cadrage à gauche.
Si

```
RSET B$=Y$
```

est spécifié, la donnée en chaîne Y$ sera placée dans la variable B$ du bloc avec cadrage à droite.
Si

```
X$="MSX"
Y$="PERSONAL COMPUTER"
```

est spécifié, la donnée sera placée dans le bloc de la façon suivante: (Remarque: A ce stade, le bloc est écrit dans le registre-tampon et il sera écrit plus tard dans le fichier sur le disque par l'énoncé PUT. On trouvera donc ici un exemple de la manière ordonnée selon laquelle les informations sont traitées dans un fichier à accès direct.)



Si la longueur de la donnée dépasse celle des variables du bloc, les caractères excédentaires sur la droite seront ignorés.

Les données de type caractère peuvent désormais être écrites dans le fichier. Mais comme seules des variables en chaîne sont utilisables dans les blocs, les données de type numérique devront encore être converties en données de type caractère avant d'être placées dans un bloc. Pour la conversion de données numériques en données en chaîne, on devra faire appel aux fonctions MKI$ (make integer dollar), MKS$ (make single precision dollar) et MKD$ (make double precision dollar).

LSET (ou RSET) variable en chaîne = MKI$ (donnée de type entier)
LSET (ou RSET) variable en chaîne = MKS$ (donnée de type simple précision)
LSET (ou RSET) variable en chaîne = MKD$ (donnée de type double précision)

```
LSET P$ = MKI$(A%)
LSET Q$ = MKS$(B!)
LSET R$ = MKD$(C#)
```

Après leur changement en données de type en chaîne, la longueur des données numériques est de 2 octets dans le cas de celles de type entier, de 4 octets pour la simple précision et de 8 octets pour la double précision. Le nombre d'octets requis pour chaque variable doit être défini dans l'énoncé FIELD en fonction de la taille des données numériques.
Dans l'exemple précédent, P$ est de type entier, Q$ à simple précision et R$ à double précision. Ces variables doivent être définies comme suit dans l'énoncé FIELD:

```
FIELD #1,2 AS P$,4 AS Q$,8 AS R$
```

(4) Sortir la donnée avec un énoncé PUT
Une fois que la donnée a été placée dans le bloc du registre tampon, l'énoncé PUT s'emploie pour l'écrire dans le fichier sur le disque.

PUT[#] numéro fichier [,numéro bloc]

L'énoncé PUT écrit la donnée, qui se trouve actuellement dans le bloc, à l'emplacement du fichier spécifié par le numéro de fichier. Ce numéro de bloc sera utilisé, par la suite, quand la donnée sera extraite du fichier à accès direct.
Si le numéro de bloc est omis dans l'énoncé PUT ou si l'énoncé GET n'a été exécuté auparavant, c'est le numéro "1" qui sera automatiquement spécifié comme numéro de bloc. Si un énoncé PUT ou GET a été exécuté au préalable, 1 sera ajouté au numéro de bloc de l'énoncé précédent pour devenir le nouveau numéro de bloc.

(5) Refermer le fichier
Le fichier est refermé par
CLOSE[#] [numéro fichier]
qui supprime le numéro affecté au fichier.

Ci-après, on trouvera un programme complet, destiné à l'écriture de données dans un fichier à accès direct.

```
10 OPEN "A:TELNO.DAT" AS #1
20 FIELD #1,2 AS ID$,12 AS NAM$,11 AS NO
$
30 FOR R=1 TO 3
40 READ A%,B$,C$
50 LSET ID$=MKI$(A%)
60 LSET NAM$=B$
70 RSET NO$=C$
80 PUT #1,R
90 NEXT R
100 CLOSE #1
110 DATA 1,TOM,111-2222
120 DATA 2,SUSIE,333-4444
130 DATA 3,JOAN,555-6666
```

Aux lignes 40 et 50, A% est une variable numérique de type entier. (Les caractères de déclaration de type %, ! et # servent à définir une variable comme de type entier, à simple précision et à double précision.)

L'exécution du programme précédent permet de créer le fichier "TELNO.DAT" sur le disque de l'unité A. La longueur totale d'un bloc pour l'entrée/sortie est de 25 octets, se ventilant comme suit: 2 octets affectés à la variable en chaîne ID$, 12 octets à NAM$ et 11 octets à NO$. Lorsque la boucle FOR—NEXT, commençant à la ligne 30, est répétée trois fois, les données suivantes sont écrites dans le fichier.

	ID$: 2 octets	NAM$: 12 octets	NO$: 11 octets
bloc 1	(1)	TOM	111-2222
bloc 2	(2)	SUSIE	333-4444
bloc 3	(3)	JOAN	555-6666

Les données de type entier 1, 2, 3 dans cette colonne sont converties en données en chaîne selon le format d'expression interne, et elles sont écrites comme 2 octets chacune.

LECTURE DE DONNEES ISSUES D'UN FICHIER A ACCES DIRECT

La marche à suivre pour l'écriture et la lecture de données dans un fichier à accès direct est pratiquement la même.
(1) Ouvrir le fichier par un énoncé OPEN.
(2) Spécifier le format d'un bloc par un énoncé FIELD.
(3) Lire la donnée issue d'un bloc par un énoncé GET.
(4) Refermer le fichier.

Les démarches (1) et (2) sont les mêmes que pour l'écriture dans un fichier, mais la lecture s'effectue par l'énoncé GET (3).

GET[#] numéro fichier, [,numéro bloc]

L'énoncé GET lit la donnée du bloc spécifié et il l'affecte à chaque variable, comme défini par l'énoncé FIELD.
A l'écriture, les données numériques avaient dû être converties en données en chaîne et, par conséquent, à la lecture, elles sont affectées aux variables en chaîne, spécifiées par l'énoncé FIELD. Pour pouvoir l'afficher sur l'écran ou la traiter, il est nécessaire de convertir à nouveau ces données en chaîne en données numériques, ce qui s'accomplit par les fonctions CVI, CVS et CVD qui remplissent exactement le rôle opposé des fonctions MKI$, MKS$ et MKD$. Si, par exemple, une donnée numérique de type entier a été affectée par l'énoncé GET à la variable en chaîne P$, elle sera changée en donnée de type entier et affectée à la variable de type entier A% par

```
A%=CVI(P$)
```

CVI(P$) peut être utilisé directement dans un énoncé PRINT pour afficher la donnée.

```
PRINT CVI(P$)
```

CVI (conversion à nombre entier) convertit la donnée en une de type entier; CVS (conversion à simple précision) convertit la donnée en une de type à simple précision, tandis que CVD (conversion à double précision) convertit la donnée en une de type à double précision.

variable de type numérique (type entier) = CVI (donnée type en chaîne)
variable de type numérique (type simple précision) = CVS (donnée type en chaîne)
variable de type numérique (type double précision) = CVD (donnée type en chaîne)

Une fois que les données ont été lues, le fichier est refermé par un énoncé CLOSE.

Ecrivons à présent un programme, afin de lire les données du fichier à accès direct que nous avions fait dans le programme précédent et de les afficher sur l'écran.

```
10 SCREEN 0:WIDTH 40
20 OPEN "A:TELNO.DAT" AS #1
30 FIELD #1,2 AS ID$,12 AS NAM$,11 AS NO
$
40 INPUT "Record number ";N
50 IF N=0 THEN GOTO 110
60 GET #1,N
70 PRINT "ID NO.";CVI(ID$);"   ";
80 PRINT NAM$;NO$
90 PRINT
100 GOTO 40
110 CLOSE #1
```

Ce programme affichera des données semblables à celles qui sont présentées sur l'illustration suivante:

```
RUN
Record number? 1
ID NO.1      TOM                 111-2222

Record number? 2
ID NO.2      SUSIE               333-4444

Record number? 0
Ok
```

Chapitre 10
Sous-programmes en langage machine

ECRITURE ET EXECUTION DE SOUS-PROGRAMMES EN LANGAGE MACHINE

- Spécification de la zone et adresse de départ—CLEAR, DEFUSR
- Ecriture d'un sous-programme en langage machine—POKE
- Appel d'un sous-programme en langage machine—USR
- Sauvegarde et chargement d'un sous-programme en langage machine—BSAVE, BLOAD

SOUS-PROGRAMMES EN LANGAGE MACHINE

Avec le langage MSX-BASIC, le langage machine de l'unité centrale de traitement Z-80 peut être utilisé pour écrire un sous-programme dans la mémoire. Le contrôle peut être transféré du BASIC au sous-programme et les résultats de l'exécution de ce dernier transmis à une variable, définie par le BASIC.
Un maximum de 10 sous-programmes en langage machine peuvent être définis et une seule valeur peut être donnée à un sous-programme en langage machine.

SPECIFICATION DE LA ZONE ET ADRESSE DE DEPART D'UN SOUS-PROGRAMME

CLEAR, DEFUSR

Pour utiliser un sous-programme en langage machine, **l'énoncé CLEAR devrait être utilisé pour spécifier l'adresse de la partie haute de mémoire de la zone programme du BASIC.** La zone suivant l'adresse supérieure sera utilisée pour l'écriture du sous-programme en langage machine.

CLEAR [taille de zone caractère] [,adresse haut de mémoire]

Par exemple,

```
CLEAR 300,&HCFFF
```

spécifie l'adresse haut de mémoire de la zone de programme BASIC comme &HCFFF (adresse décimale 53427). Par conséquent, la zone commençant par l'adresse &HD000 (addresse décimale 53248) est utilisable comme zone d'écriture d'un sous-programme en langage machine. Dans l'énoncé CLEAR ci-dessus, la taille de la zone caractère est spécifiée comme étant de 300 octets. Le réglage implicite initial est de 400 octets.

Ensuite, l'adresse de départ de sous-programme est définie par l'énoncé DEFUSR.

DEFUSR[X] = adresse départ

X est un nombre entier de 0 à 9. L'adresse départ de 10 sous-programmes peut être définie par la fonction USR.

```
DEFUSR0=&HD000
```

Cet énoncé DEFUSR définit le lancement du sous-programme à l'adresse &HD000 (adresse décimale 53248) comme la fonction USR 0.

ECRITURE D'UN SOUS-PROGRAMME EN LANGAGE MACHINE POKE

L'énoncé POKE s'emploie pour écrire un sous-programme en langage machine dans la mémoire.

POKE adresse, expression

L'énoncé POKE écrit un octet de donnée dans l'adresse spécifiée de la mémoire.
Le programme suivant écrit les nombres hexadécimaux 21, 3C, F0, C9 dans la mémoire, en commençant à l'adresse &HD000.

```
100 AD=&HD000
110 READ M$:IF M$="END" THEN END
120 POKE AD,VAL("&H"+M$)
130 AD=AD+1:GOTO 110
140 DATA 21,3C,F0,C9
150 DATA END
```

Il suffit d'écrire les instructions du sous-programme en langage machine (instructions du Z-80) dans l'énoncé DATA de la ligne 140. L'instruction RET ramène le contrôle, du sous-programme en langage machine, vers le programme BASIC.

APPEL D'UN SOUS-PROGRAMME EN LANGAGE MACHINE USR

Le transfert du contrôle à un sous-programme en langage machine porte le nom d'appel de sous-programme en langage machine.

La fonction USR s'emploie pour l'appel d'un tel sous-programme.

USR[X](I)

X est le numéro de la fonction USR, défini par l'énoncé DEFUSR. I est la valeur (numérique ou en chaîne) donnée au sous-programme en langage machine.
Par exemple, le sous-programme défini par DEFUSR 0 est appelé en exécutant

```
X=USR0(Y)
```

Après que le sous-programme en langage machine est exécuté, la valeur résultant de l'exécution est affectée á la variable X et le programme BASIC est exécuté.
Quand le sous-programme en langage machine est appelé, la valeur donnée au sous-programme (valeur de variable Y) est stockée dans les emplacements suivants de la mémoire, et les données qui indiquent le type de Y sont entrées dans le registre A. L'adresse départ pour la zone qui stocke la valeur Y est entrée dans les registres HL.

Type Y	Données entrées en registre A*	Indication adresse des registres HL	Adresse de stockage de valeur Y
entier	2	&HF7F6	&HF7FB—&HF7F9
simple précision	4		&HF7F6—&HF7F9
double précision	8		&HF7F6—&HF7FD

* la même donnée est aussi entrée dans l'adresse &HF663 en mémoire.

Quand Y est une variable en chaîne:

Données entrées en registre A	Données entrées en registres DE	Descripteur de chaîne
3	Adresse de départ de descripteur de chaîne	1er octet: longueur de chaîne caractère 2ème & 3 ème octets: adresse départ de zone de mémoire de chaîne de caractère

Quand l'exécution de sous-programme en langage machine est terminée, la valeur du résultat est rendue à la variable X en réglant comme suit les registres et la mémoire:

Type valeur résultat	Adresse mémoire &HF663	Registres DE	Registres HL	Adresse stockage de résultat
entier	2		&HF7F6	&HF7F8—&HF7F9
simple précision	4		&HF7F6	&HF7F6—&HF7F9
double précision	8		&HF7F6	&HF7F6—&HF7FD
chaîne	3	adresse de départ de descripteur de chaîne		zone commençant à l'indicateur d'adresse par 2ème et 3ème octets de descripteur de chaîne

Le nombre généré qui doit être renvoyé au BASIC est mémorisé aux adresses &HF7F8 et &HF7F9. En outre, comme le type de valeur est un nombre entier, on entrera un 2 dans l'adresse &HF663.
Le programme BASIC suivant écrira le sous-programme en langage machine dans la mémoire et, il le rappellera et utilisera les valeurs fournies (valeur aléatoire de 0 à 255) pour afficher un chiffre de 1 à 6 sur le dé.

```
10 CLEAR 300,&HCFFF ——spécifie l'adresse haut de mémoire
20 SET BEEP 1          de la zone de programme BASIC
30 '--
40 AD=&HD000:DEFUSR0=AD————fixe l'adresse départ de
50 GOSUB 430               sous-programme à &HD000
60 ' --
70 SCREEN 5
80 OPEN "GRP:" FOR OUTPUT AS #1
90 SET PAGE 1,1:CLS
100 R=5:RESTORE 360
110 FOR L=0 TO 5
120 XC=(L MOD 3)*80
130 YC=(L \ 3)*100                 trace les points
140 LINE (XC,YC)-STEP(50,50),15,BF (1 à 6) sur le
150 FOR M=0 TO L                   dé en page 1
160 READ X,Y
170 CIRCLE (XC+X,YC+Y),R,8
180 PAINT (XC+X,YC+Y),8,8
190 NEXT M
200 NEXT L
210 ' --
220 SET PAGE 0,0
230 PRESET (70,150)             appelle le sous-programme
240 PRINT #1,"Press any key"    quand une touche est
250 IF INKEY$="" THEN 250———————actionnée
260 J=USR0(0) MOD 20————————————appel de sous-programme
270 FOR L=0 TO J
280 N=USR0(0) MOD 6—————————————appel de sous-programme
290 X=(N MOD 3)*80                             copie les
300 Y=(N \ 3)*100                              points (1 à 6)
310 COPY (X,Y)-STEP(60,60),1 TO (100,70)       du dé sur la
,0                                             page 1 en
320 BEEP:FOR W=0 TO 50:NEXT W                  fonction de
330 NEXT L                                     la valeur
340 GOTO 250                                   fournie par
350 ' *** dice data ***                        le sous-
360 DATA 25,25                                 programme
370 DATA 25,10,25,40
380 DATA 10,10,25,25,40,40
390 DATA 10,10,10,40,40,10,40,40               donnée
400 DATA 10,10,10,40,40,10,40,40,25,25         déterminant
410 DATA 10,10,10,40,40,10,40,40,10,25,4       l'emplace-
0,25                                           ment des
                                               points du dé
```

```
420 ' *** write machine language subrout
ine ***
430 RESTORE 480
440 READ M$:IF M$="END" THEN RETURN
450 POKE AD,VAL("&H"+M$)
460 AD=AD+1:GOTO 440
470 ' *** machine codes ***
480 DATA 21,F8,F7,ED,5F,77,23,AF
490 DATA 77,3E,02,32,63,F6,C9
500 DATA END
```

écrit le sous-programme en langage machine

Exemple d'un sous-programme en langage machine

Le programme suivant produit des nombres aléatoires de 0 à 255 en utilisant le registre R (registre de regénération) de l'unité centrale de traitement. On trouvera ci-dessous la liste d'origine. (Le format est basé sur MACRO80 3.44).

```
        MACRO-80 3.44   09-Dec-81       PAGE    1


                                        .Z80
                                        .PHASE  0D000H

 0002                           INTEGER EQU     2
 F663                           VALTYPE EQU     0F663H
 F7F6                           DAC     EQU     0F7F6H

                                ;
                                ;--     machine-language sample program
                                ;

 D000                           START:
 D000     21 F7F8                       LD      HL,DAC+2
 D003     ED 5F                         LD      A,R             ; load R register
 D005     77                            LD      (HL),A
 D006     23                            INC     HL
 D007     AF                            XOR     A
 D008     77                            LD      (HL),A          ; set random data
 D009     3E 02                         LD      A,INTEGER
 D00B     32 F663                       LD      (VALTYPE),A     ; set data type
 D00E     C9                            RET

                                        END     START
        MACRO-80 3.44   09-Dec-81       PAGE    S


Macros:

Symbols:
F7F6    DAC             0002    INTEGER         D000    START
F663    VALTYPE



No Fatal error(s)


```

SAUVEGARDE D'UN SOUS-PROGRAMME EN LANGAGE MACHINE BSAVE

Le sous-programme en langage machine ci-dessus est écrit dans les adresses &HD000—&HD00E en mémoire.
L'instruction BSAVE est utilisée pour sauvegarder ce sous-programme sur une cassette ou sur un disque.

BSAVE "[nom périphérique] [nom fichier [.nom type]]", adresse départ, adresse fin [,adresse départ d'exécution]

L'instruction BSAVE sauvegarde le contenu sur la zone spécifiée en mémoire.

L'énoncé

```
BSAVE "CAS:RANDOM",&HD000,&HD00E
```

sauvegarde le contenu des adresses &HD000—&HD00E (programme en langage machine) sur une cassette sous le nom de fichier RANDOM.

L'énoncé

```
BSAVE "A:RANDOM.BIN",&HD000,&HD00E
```

sauvegarde le même contenu sur une disquette placée dans l'unité A sous le nom de fichier/type RANDOM.BIN.

CHARGEMENT D'UN SOUS-PROGRAMME EN LANGAGE MACHINE BLOAD

Le contenu sauvegardé par l'instruction BSAVE est chargé par l'instruction BLOAD.

BLOAD "[nom périphérique] [nom fichier [.nom type]]" [,R] [,offset]

L'énoncé

```
BLOAD "CAS:RANDOM"
```

charge le contenu du fichier sauvegardé sur la cassette sous le nom RANDOM.

L'énoncé

```
BLOAD "A:RANDOM.BIN"
```

charge le contenu du fichier sauvegardé sur le disque de l'unité A sous le nom de fichier/type RANDOM.BIN.

Dans les deux cas, le contenu de la zone, commençant à l'adresse départ spécifiée dans l'instruction BSAVE, est chargé.

Lorsqu'un sous-programme en langage machine est sauvegardé séparément de la manière précédente, il peut être chargé et exécuté dans un programme BASIC sans écrire le sous-programme par l'énoncé POKE. Dans ce cas, la ligne 50 et les lignes 420 à 500 ne seront pas requises dans le programme BASIC ci-dessus.

INDEX

INSTRUCTIONS, ENONCES, FONCTIONS ET MESSAGES D'ERREUR DU BASIC

TERMES UTILISES

UTILISATION DU BASIC

En mode graphique
Définition de l'analyse entrelacée (SCREEN) ... 152
Vidage de l'écran (CLS) ... 41
Changement des pages (SET PAGE) ... 123
Copiage des données de figure (COPY) ... 127
Copiage avec emploi d'opération logique ... 141

Couleur
Définition de la palette couleur (COLOR) ... 108
Définition des codes couleur en mode SCREEN 8 ... 113

Lutins (Sprite)
Définition d'un motif sprite ... 159
Affichage d'un motif sprite (PUT SPRITE) ... 163
Animation d'un lutin ... 165
Changement de la couleur d'un lutin (COLOR SPRITE) ... 168
Spécification de la couleur de chaque ligne d'un lutin (COLOR SPRITE$) ... 170
Déplacement de 32 points d'une ligne de lutin ... 172

Technique de programmation
Technique de spécification des motifs sprite ... 174
Obtention d'un nombre aléatoire différent à chaque exécution ... 191
Avance du programme par poussée sur une touche ... 207
Avance du programme par poussée sur une touche spécifique ... 208
Affichage des caractères en mode graphique ... 250

GUIDE DU MSX-BASIC Version 2.0

Sony Corporation Printed in Japan © 1985 3-764-318-32 (1)